

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau



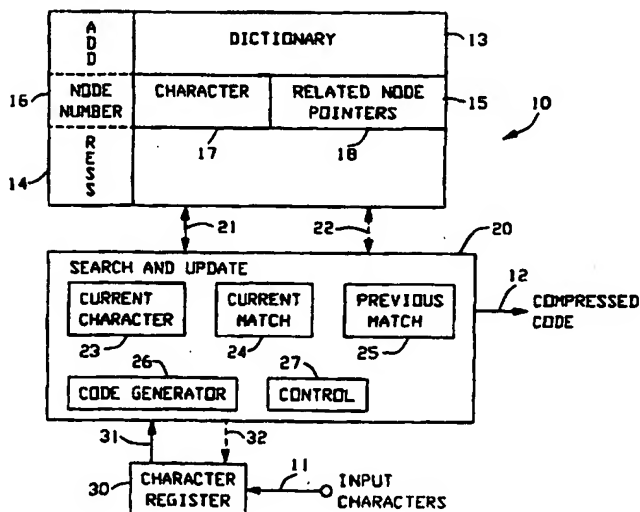
INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H03M 7/30		A1	(11) International Publication Number: WO 98/04045
			(43) International Publication Date: 29 January 1998 (29.01.98)
(21) International Application Number: PCT/US97/12943		(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).	
(22) International Filing Date: 23 July 1997 (23.07.97)			
(30) Priority Data: 60/023,094 24 July 1996 (24.07.96) US 08/753,871 3 December 1996 (03.12.96) US			
(71) Applicant: UNISYS CORPORATION [US/US]; Township Line and Union Meeting Roads, MS C1SW19, Blue Bell, PA 19424-0001 (US).			
(72) Inventors: WELCH, Terry, A.; 9202 Mystic Oaks Trail, Austin, TX 78750 (US). COOPER, Albert, B.; 20 East 74th Street #14-C, New York, NY 10021 (US).			
(74) Agent: STARR, Mark, T.; Unisys Corporation, Township Line & Union Meeting Roads, M/S C1SW19, Blue Bell, PA 19424-0001 (US).		Published With international search report. Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.	

(54) Title: DATA COMPRESSION AND DECOMPRESSION SYSTEM WITH IMMEDIATE DICTIONARY UPDATING INTERLEAVED WITH STRING SEARCH

(57) Abstract

A dictionary based data compression and decompression system where, in the compressor (10), when a partial string W and a character C are matched in the dictionary (13), a new string is entered into the dictionary with C as an extension character on the string PW where P is the string corresponding to the last output compressed code signal. An update string is entered (113) into the compression dictionary for each input character that is read and matched. The updating is immediate and interleaved with the character-by-character matching of the current string. The update process continues until the longest match is found in the dictionary. The code of the longest matched string is output (106) in a string matching cycle. If a single character or multi-character string "A" exists in the dictionary, the string AAA...A is encoded in two compressed code signals regardless of the string length. This encoding results in an unrecognized code signal at the decompressor. The decompressor (40), in response to an unrecognized code signal, enters (Fig. 8) update strings into the decompressor dictionary (43) in accordance with the recovered string (161) corresponding to the previously received code signal, the unrecognized code signal, the extant code of the decompressor and the number (135) of characters in the previously recovered string.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LJ	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

DATA COMPRESSION AND DECOMPRESSION SYSTEM WITH IMMEDIATE DICTIONARY
UPDATING INTERLEAVED WITH STRING SEARCHBACKGROUND OF THE INVENTION1 1. Field of the Invention

 The invention relates to dictionary based data
compression and decompression particularly with respect
to the manner in which the compression and decompression
5 dictionaries are updated.

2. Description of the Prior Art

 The Lempel-Ziv (LZ) algorithm known as LZ2
provides the theoretical basis for numerous dictionary
based data compression and decompression systems in
widespread usage. LZ2 is described in a paper entitled
10 "Compression Of Individual Sequences Via Variable-Rate
Coding" by Jacob Ziv and Abraham Lempel, published in
the IEEE Transactions on Information Theory, Vol. IT-24,
No. 5, September 1978, pages 530-536. A ubiquitously
15 used data compression and decompression system known
as LZW, adopted as the standard for V.42 bis modem
compression and decompression, is described in U.S. Patent
4,558,302 by Welch, issued December 10, 1985. LZW has
also been adopted as the compression and decompression
20 methods used in the GIF and TIFF image communication
standards. A variant of LZ2 is described in U.S. Patent
4,876,541 by Storer, issued October 24, 1989. Further
examples of LZ dictionary based compression and
decompression systems are described in U.S. Patent
25 4,464,650 by Eastman et al., issued August 7, 1984; U.S.
Patent 4,814,746 by Miller et al., issued March 21, 1989;
U.S. Patent 5,153,591 by Clark, issued October 6, 1992;
and European Patent Application Publication Number 0
573 208 A1 by Lempel et al., published December 8, 1993.

30 In the above-cited systems, the input data
character stream is compared character-by-character with

- 2 -

1 character strings stored in a dictionary to effect a
match therewith. Typically, the character-by-character
comparison is continued until the longest match is
determined. Based on the match, a compressed code is
5 output and the dictionary is updated with one or more
additional character strings. In the Storer patent ('541)
the dictionary is updated by concatenating all of the
non-zero prefixes of the current longest matched string
with the previous longest matched string. Thus, if there
10 are N characters in the current longest match, N strings
are added to the dictionary after the current longest
match is determined. In the Storer patent this is denoted
as the All Prefixes (AP) update technique.

Another type of data compression and decompression
15 method is denoted as Run-Length Encoding (RLE). The
RLE algorithm compresses a repeating character or
character group run by providing a compressed code
indicating the character or character group and the length
of the run. RLE is thus effective in encoding long runs
20 of the same character or group of characters. For
example, RLE is effective in compressing a long sequence
of blanks that may be included at the beginning of a
data file. RLE is also effective in image compression
where an image contains a long run of consecutive pixels
25 having the same value, such as in the sky portion of
a land-sky image.

The LZ dictionary based compression and
decompression algorithms discussed above are not
especially effective in compressing long runs of a
30 repeating character or character group. Even utilizing
the AP update technique, a large number of compressed
code outputs are required to compress a long length run.

This deficiency of the dictionary based systems
is traditionally overcome by applying the data to a run
35 length encoder and applying the run length encoded data
to the LZ dictionary based system. In such an
architecture a run length encoder is utilized at the

1 front end of the dictionary based compressor and a run
length decoder is utilized at the output end of the
dictionary based decompressor. Such a system suffers
from the disadvantages of increased equipment, expense,
5 control overhead and processing time.

SUMMARY OF THE INVENTION

The invention is embodied in a dictionary based
data compression and decompression system that overcomes
10 the above-described deficiency. If a string A exists
in the dictionary, then the string AAA...A is encoded
in two compressed code symbols, regardless of its length.
Thus, strings of repeated characters, such as blanks
and zeros, or character groups such as consecutive image
15 pixels with the same value, can be encoded very
efficiently on the first encounter.

In the compression algorithm of the present
invention a string is entered into the compression
dictionary as each input character is read and matched.
20 Conventionally, an update string or strings is entered
into the dictionary when a longest match is achieved
and an output compressed code symbol is determined.
The operation of the algorithm is as follows. Each time
a partial string W and character C are found in the
25 dictionary, a new string is entered into the dictionary
with C as an extension character on the string PW where
P was the string conveyed in the last transmitted output
compressed code symbol. Thus, as the string W is matched,
the string P is extended by the characters of W as they
30 are matched in the string searching process. This might
be referred to as "on-the-fly" dictionary updating where
the dictionary updating is immediate and interleaved
character-by-character with the string searching process.
Thus, as a character-by-character match of the input
35 with the stored string W is effected, each matching
character is appended to the end of the growing string
PW. The update process ends when the input data

1 characters have matched the longest string W in the dictionary.

When the string W being matched coincides with the previously matched string P, the run length encoding advantage described above is realized. When this occurs, the compressor transmits a compressed code symbol that is unrecognized at the decompressor. The decompressor uses an unrecognized code process to maintain synchronism with the compression dictionary, the process based on the currently assigned decompressor code, the unrecognized code, the previously decoded string and the number of characters in the previously decoded string.

BRIEF DESCRIPTION OF THE DRAWINGS

15 Figure 1 is a schematic block diagram of a data compression subsystem used in embodying the present invention.

Figure 2 is a schematic block diagram of a data decompression subsystem for recovering the compressed code output of the compressor of Figure 1.

20 Figure 3a is a diagram illustrating a representational data structure for the nodes of the searchtrees of the dictionaries of Figures 1 and 2.

Figure 3b is a diagram illustrating a practical data structure for the nodes of the searchtrees of the dictionaries of Figures 1 and 2.

25 Figure 4 is a schematic node diagram illustrating a node of the searchtrees of the dictionaries of Figures 1 and 2 in accordance with the data structure of Figure 3a.

30 Figure 4a is a schematic representation of a partial searchtree illustrating data storage utilizing the node of Figure 4.

Figure 5 is a schematic node diagram illustrating a node of the searchtrees of the dictionaries of Figures 1 and 2 in accordance with the data structure of Figure 3b.

1 Figure 5a is a schematic representation of a
partial searchtree utilizing the node of Figure 5 and
storing the same strings as Figure 4a.

5 Figure 6 is a control flow chart illustrating
the operations executed by the compression subsystem
of Figure 1 so as to perform data compression in
accordance with the present invention. The flow chart
of Figure 6 is predicated on a compression dictionary
initialized with all single character strings.

10 Figure 7 is a control flow chart illustrating
the operations executed by the decompression subsystem
of Figure 2 for decompressing the compressed code
generated in accordance with Figure 6. The flow chart
of Figure 7 is predicated on a decompression dictionary
15 initialized with all single character strings.

Figure 8 is a control flow chart illustrating
the unrecognized code processing of Figures 7 and 10.

Figure 9 is a control flow chart similar to that
of Figure 6 but predicated on a non-initialized
20 compression dictionary.

Figure 10 is a control flow chart similar to
that of Figure 7 but predicated on a non-initialized
decompression dictionary. The decompression flow chart
of Figure 10 decompresses the compressed code generated
25 in accordance with Figure 9.

Figures 11a-11e are schematic representations
of partial searchtrees illustrating consecutive states
of the compression dictionary when compressing a typical
input data character stream.

30 Figures 12a-12g are schematic representations
of partial searchtrees illustrating consecutive states
of the compression dictionary when the input data
character stream is a repeating character group.

35 DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to Figure 1, a data compression
subsystem 10 is illustrated that compresses a stream

- 6 -

1 of input data character signals applied at an input 11
into a stream of corresponding compressed code signals
at an output 12. A dictionary 13 for storing character
strings is implemented by a memory such as a RAM or CAM
5 generally in the manner described in the above-cited
references. The character strings are stored in a
searchtree database structure in a manner that is well
understood. The searchtree is comprised of interlinked
nodes stored at the locations of the dictionary 13.
10 The memory locations of the dictionary 13 are accessed
by addresses 14 in a well known manner.

The data structure of a searchtree node is
illustrated by a node 15 which includes a node number
16, a character field 17 and fields 18 for related node
15 pointers. The node number 16 identifies the tree node
and the memory address 14, at which the node 15 is stored,
is utilized as the node number for convenience. The
character field 17 is utilized to contain the data
character value of the node. Fields 18 contain pointers
20 that link the node 15 to related tree nodes such as
parent, children and sibling nodes in a well understood
manner.

The compression subsystem 10 includes a search
and update control section 20 coupled to the dictionary
25 13 via a bi-directional data bus 21 and a bi-directional
control bus 22. The search and update control section
20 includes working registers denoted as a current
character register 23, a current match register 24 and
a previous match register 25. The search and update
30 control section 20 further includes a code generator
26 for assigning compressed code values to character
strings stored in the dictionary 13. The code generator
26 may assign code numbers sequentially or pseudorandomly
such as by hashing. The assigned codes access the
35 locations of the dictionary 13 via the memory addresses
14. Thus, as is well understood, the addresses 14 (node
numbers 16) are utilized as the compressed codes for

- 7 -

1 the strings stored in the dictionary 13.

The search and update control section 20 includes control 27 for controlling the operations of the compression subsystem 10 in accordance with the operational flow charts of Figures 6 and 9 in a manner to be described.

The compression subsystem 10 includes a character register 30 that buffers the input data character stream received at the input 11. The individual input data characters are applied from the character register 30 via a bus 31 to the current character register 23 in accordance with operations to be described. The search and update control section 20 controls acquiring input data characters from the character register 30 via a control bus 32.

The operation of the compression subsystem 10 is briefly as follows. Input data characters are sequentially inserted into the current character register 23 and searched against the strings stored in the dictionary 13 until the longest match therewith is achieved. The current match register 24 is utilized in this process. The node number 16 of the longest matched string is provided as the compressed code at the output 12. These searching operations are the same as those described in the above-cited references. In accordance with the invention, as the input data characters match the characters of the stored dictionary string being searched, the dictionary 13 is updated by extending the longest string matched in the previous iteration by the current input characters as they are matched. The previous match register 25 is utilized in this process. The previous matched string, so extended, is available for matching as the input characters continue to be fetched and matched. Thus, update strings are immediately added to the dictionary 13 in an interleaved manner with respect to the character-by-character string search.

- 8 -

1 Referring to Figure 2 with continued reference
to Figure 1, a decompression subsystem 40 is illustrated
that recovers the characters of the original input data
stream from the compressed code signals provided at the
5 output 12 of the compression subsystem 10. Accordingly,
the decompression subsystem 40 receives input compressed
code signals at an input 41 and provides the corresponding
recovered string characters at an output 42. The
decompression subsystem 40 includes a dictionary 43 that
10 is preferably implemented by RAM memory. The dictionary
43 is structured and updated to contain the same
searchtree database as contained in the dictionary 13
of the compression subsystem 10. As each input compressed
code is received at the input 41, the dictionary 43 is
15 updated to contain the same data character strings stored
in the dictionary 13. The searchtree database structure
stored in the dictionary 43 is comprised of interlinked
nodes stored at the locations of the dictionary 43.
The memory locations of the dictionary 43 are accessed
20 by addresses 44 in a well-known manner.

The data structure of a searchtree node is
illustrated by a node 45 which, as described above with
respect to the node 15 of the dictionary 13, includes
a node number 46, a character field 47 and fields 48
25 for related node pointers. As described above with
respect to the dictionary 13, the node number 46
identifies the tree node and the memory address 44, at
which the node 45 is stored, is utilized as the node
number. The character field 47 is utilized to contain
30 the data character value of the node. Fields 48 contain
pointers that link the node 45 to related tree nodes
such as parent, children and sibling nodes as described
above.

The decompression subsystem 40 includes a recover
35 and update control section 50 coupled to the dictionary
43 via a bi-directional data bus 51 and a bi-directional
control bus 52. The recover and update control section

1 50 includes working registers denoted as a current
received code register 53 and a previous string register
54. In accordance with the invention, the recover and
update control section 50 includes an unrecognized code
5 processing section 55 to be explained in detail with
respect to Figure 8.

The recover and update control section 50 further
includes a code generator 56 for assigning compressed
code values to character strings stored in the dictionary
10 43. The code generator 56 may assign code numbers
sequentially or pseudorandomly such as by hashing. For
system compatibility, the code generator 56 assigns code
numbers utilizing the same process and algorithm utilized
by the code generator 26 of the compression subsystem
15 10. The assigned codes access the locations of the
dictionary 43 via the memory addresses 44. Thus, as
described above with respect to the compression subsystem
10, the addresses 44 (node numbers 46) are utilized as
the codes for the strings stored in the dictionary 43.

20 The recover and update control section 50 includes
control 57 for controlling the operations of the
decompression subsystem 40 in accordance with the
operational flow charts of Figures 7, 8 and 10 in a manner
to be described.

25 The decompression subsystem 40 includes a code
register 60 that buffers the compressed code signals
received at the input 41. The individual compressed
code signals are applied from the code register 60 via
a bus 61 to the current received code register 53 in
30 accordance with operations to be described. The recover
and update control section 50 controls acquiring
compressed code signals from the code register 60 via
a control bus 62.

The operation of the decompression subsystem
35 40 is briefly as follows. An input compressed code signal
inserted into the current received code register 53
accesses the corresponding string stored in the

1 dictionary 43 via the addresses 44. The characters of
the string are recovered from character field 47 as the
recovery process traces the nodes of the string backward
through the searchtree utilizing the related node pointers
5 48. The recovered characters of the string are provided
in the appropriate order at the output 42. These string
recovery operations are the same as those described in
the above-cited references. The dictionary 43 is updated
by extending the previously recovered string by each
10 character of the currently recovered string. The previous
string register 54 is utilized in this process.

An unrecognized compressed code signal that does
not have a corresponding string stored in the dictionary
43 will be received in response to the compression
15 subsystem 10 compressing a repeated character or character
group string. When the unrecognized compressed code
signal is received, the unrecognized code processing
55 is utilized to recover the string corresponding to
the unrecognized compressed code signal. In addition,
20 update strings corresponding to those that were stored
in the dictionary 13 of the compression subsystem 10
during this compression process are also stored in the
dictionary 43 of the decompression subsystem 40. Details
of the unrecognized code processing 55 will be explained
25 below with respect to Figure 8.

Referring to Figure 3a, a representational data
structure for the nodes of the searchtrees of the
dictionaries 13 and 43 is illustrated. Since, in the
preferred embodiments of the invention, the same node
30 data structure is utilized in both the compression
subsystem 10 and the decompression subsystem 40, common
reference numerals from both Figures 1 and 2 are shown
in Figure 3a. The node number 16,46 and character field
17,47 were explained above. The related node pointer
35 fields 18,48 include a parent pointer field 66 and
children pointer fields 67. In a well-known manner,
the parent pointer field 66 contains the node number

- 11 -

1 of the parent node of the current node 15,45 and the
children pointer fields 67 contain the node numbers of
the children nodes of the current node 15,45.

5 In a manner well appreciated in the art, the
compression subsystem 10 effects a downward search through
the searchtree as follows. When residing at a current
node, the character value of the children nodes are
examined to determine if any child node matches the
current input character. If a match occurs, the child
10 node becomes the current node and the process is repeated
with the next input character until a current node is
encountered that does not have a child node that matches
the current input character. When this occurs, the
longest matching string has been found in the dictionary
15 13 and the node number thereof is used as the compressed
code signal for this longest matched string. A forward
search through the searchtree begins at a root node
whereat the parent pointer field 66 would contain a null
value.

20 In an equivalent manner, as is known, the forward
search can be performed by finding the next node to search
from a hashing function of the current node number and
the current input character. In such an embodiment,
the children pointer fields 67 would not be utilized.
25 The Welch patent ('302) discloses hash search embodiments
of the LZW algorithm.

In a well-known manner, the data structure of
Figure 3a is also used by the decompression subsystem
40 in a backward search through the searchtree to recover
30 a data character string corresponding to a compressed
code signal. The compressed code signal addresses the
node number 46 and the character value in the character
field 47 is stored. The node number in the parent pointer
field 66 is then utilized to access the parent node and
35 the character value therein is stored. The process is
continued until the root node is attained. Since the
characters are recovered by this process in reverse order,

1 a mechanism such as a LIFO stack or an appropriately
configured output buffer is utilized to reverse the
character order thereby recovering the original data
character string.

5 A string stored in the compression dictionary
13 or in the decompression dictionary 43 is extended
as follows. A next available code of an empty location
is provided by the code generator 26 or 56 and the node
number thereof is added to the children pointers 67 of
10 the node being extended. The node number of the node
being extended is inserted in the parent pointer field
66 of the empty location. The character value of the
extension character is inserted in the character field
17 or 47 of the empty location.

15 Referring to Figure 3b, a practical data structure
for the nodes of the searchtrees of the dictionaries
13 and 43 is illustrated. This data structure and its
implementations in a compression and decompression system
are described in the Clark patent ('591). As explained
20 with respect to Figure 3a, the data structure of Figure
3b may be utilized in both the compression dictionary
13 and the decompression dictionary 43 and thus common
reference numerals from both Figures 1 and 2 are shown.
Again, the node number 16,46 and character field 17,47
25 are explained above. In the data structure of Figure
3b the related node pointer fields 18,48 comprise a parent
pointer field 70, a child pointer field 71 and a sibling
pointer field 72. The parent pointer field 70 is utilized
in the same manner as described above with respect to
30 the parent pointer field 66 of Figure 3a. The child
pointer field 71 and the sibling pointer field 72 replace
the children pointer fields 67 of Figure 3a. In the
data structure of Figure 3b a parent node uses its child
pointer field 71 to point to the node number of one of
35 its children nodes and the pointed to child node uses
its sibling pointer field 72 to point to the node number
of one of its sibling nodes. The pointed to sibling

- 13 -

1 node, in turn, uses its sibling pointer field 72 to point to a further sibling node. In this manner, the pointers for all of the children of a parent node are contained in a linked list of sibling nodes.

5 A downward search is performed in the manner described above with respect to Figure 3a, except that the presence of an input character is searched in the list of siblings. A backward search for the purposes of string recovery is performed in the manner described
10 above with respect to Figure 3a and implemented by setting the parent pointer field of the child and all of its siblings to the node number of the parent. In order to facilitate searching, the sibling list may be arranged in the order of ascending character value.

15 A string represented by a childless node (child pointer field = 0) is extended by assigning the next available code, as described above, designating the next available empty location and inserting the node number of this empty location into the child pointer field 71
20 of the node being extended. The parent pointer field 70 of this newly created child node is set to the node number of the parent and the extension character value is inserted into the character field of the newly created child. If the node to be extended already has a child,
25 a new sibling node is created and inserted into the sibling list by adjusting the sibling pointer fields 72 of the appropriate nodes of the sibling list. The node number of the parent is inserted into the parent pointer field 70 of the newly created sibling node.

30 Referring to Figures 4 and 4a, Figure 4 schematically illustrates a searchtree node 80 in accordance with the data structure of Figure 3a. The address (node number), character value, parent node and children nodes are as indicated by the legends. Figure
35 4a is a representation of a partial searchtree illustrating data storage utilizing the arrangement of node 80 of Figure 4. The partial searchtree of Figure 4a

- 14 -

1 is comprised of nodes 81, 82, 83 and 84 storing the
strings ab, ac and ad. Thus, the children pointer fields
67 (Figure 3a) of parent node 81 will contain the node
numbers of children nodes 82, 83 and 84, while the parent
5 pointer field 66 of the children nodes 82, 83 and 84
will each contain the node number of parent node 81.

Referring to Figures 5 and 5a, Figure 5
schematically illustrates a searchtree node 90 in
accordance with the data structure of Figure 3b. The
10 address (node number), character value, parent node,
child node and sibling node are as indicated by the
legends. Figure 5a is a representation of a partial
searchtree utilizing the arrangement of node 90 of Figure
5 and is comprised of nodes 91, 92, 93 and 94. The
15 partial searchtree of Figure 5a stores the same strings
as that of Figure 4a. Thus, the child pointer field
71 (Figure 3b) of the parent node 91 is set to the node
number of child node 92. The sibling pointer field 72
of the child node 92 is set to the node number of sibling
20 node 93 and the sibling pointer field 72 of the sibling
node 93 is set to the node number of the sibling node
94. The parent pointer field 70 of the children nodes
92, 93 and 94 are each set to the node number of the
parent node 91.

25 In the detailed descriptions of Figures 6-10
to follow, the operations will be explained in terms
of the data structure of Figure 3b, the corresponding
node arrangement of Figure 5 and the corresponding
searchtree structure of Figure 5a. Codes will be
30 considered as assigned sequentially by the code generators
26 and 56 although it is appreciated that codes may be
assigned pseudorandomly such as by hashing. In a hashing
embodiment, a node number code and a character are hashed
to determine a next following address. In such a hashing
35 embodiment the child and sibling pointers would not be
utilized. In the flow charts of Figures 6-10, however,
the operating blocks CODE = NEXT AVAILABLE CODE encompass

1 either a next sequential code or a next hashed code.
In the sequential code assignment embodiments, these
operational blocks will be more specifically CODE=CODE+1.

Referring to Figure 6, with continued reference
5 to Figures 1 and 3b, a control flow chart is illustrated
showing the detailed operations to be executed by the
search and update control section 20 so as to perform
data compression in accordance with the invention.
Control 27 is considered as containing appropriate
10 circuitry such as state machines to control execution
of the operations.

The flow chart of Figure 6 is predicated on the
compression dictionary 13 initialized with all single
character strings. Accordingly, a block 100 provides
15 for clearing and initializing the dictionary 13 with
all of the single character strings stored at respective
codes (node numbers). This operation is performed using
the code generator 26 which sequentially assigns the
node numbers for storing the single character strings.
20 In an ASCII implementation, the first 256 codes will
be assigned by the code generator 26 for storing the
256 single character strings. Initialization is effected
by setting the character field 17 of the initialized
memory locations to the character value of the respective
25 characters of the alphabet over which the compression
is occurring. The parent pointer field 70, child pointer
field 71 and sibling pointer field 72 of these initialized
locations are set to zero. It is appreciated that the
initialized locations provide the root nodes for storing
30 strings in the dictionary 13 and thus, the parent pointer
fields 70 of these initialized locations will remain
at zero.

By these operations, the initial locations of
the dictionary 13 are set to contain the respective single
35 character strings. In an ASCII embodiment, the first
256 locations of the dictionary 13 contain the respective
256 single character strings. In the operations of the

1 block 100, the remaining locations of the dictionary
13 are cleared by setting all of the fields thereof to
zero. In an ASCII implementation the dictionary locations
with node numbers of 266 and greater are cleared.

5 At a block 101, the current match register 24
is set to zero and at a block 102 the previous match
register 25 is set to zero. At a block 103, the next
input character is placed into the current character
register 23.

10 At a block 104, a search is made to determine
if the current matched string concatenated by the current
character is in the dictionary. Any known appropriate
dictionary searching procedure, such as those described
in the above-cited references, may be utilized.

15 Specifically herein, for a non-zero current match, the
current match register 24 contains the node number of
the current matched string. The child pointed to by
the child pointer field 71 of the current match node
is compared with the input character. If the input
20 character matches the child of the current node, then
the decision of block 104 is answered in the affirmative
and the YES path is taken. If the child node does not
match the current character, the sibling list pointed
to by the child node is inspected to determine if the
25 current character matches a sibling. If a match is found,
the YES path is taken. If, however, the current node
is childless or the current character does not match
any child of the current node, the NO path from the block
104 is taken.

30 If the current match is zero, the block 104 is
seeking a root node in the dictionary 13 having the
character value equal to the current character. Since
the dictionary 13 is initialized with all single character
strings, the YES branch from the block 104 is
35 automatically taken.

When the YES branch from the block 104 is taken,
the compression processing of Figure 6 is at a point

- 17 -

1 where the current match concatenated by the current
character has been found in the dictionary 13 and the
search for a still longer string will be continued.
Accordingly, the current match is updated in a block
5 105 so that the new current match is set equal to the
existing current match concatenated by the current
character. This is accomplished by appropriately updating
the node number in the current match register 24. When
the current match is non-zero, the current match register
10 24 is updated with the node number of the child node
(or sibling node) that matched the current character
as discussed above. If the current match is zero, the
current match register 24 is updated with the node number
of the single character string that matched the current
15 character. The single character node number can be
obtained algorithmically in a well-known manner, or can
be found by searching the initialized locations for the
current character value.

If the NO branch is taken from the block 104,
20 the current match concatenated by the current character
does not match a string stored in the dictionary 13.
The current match, that had been found in the dictionary,
provides the longest match with the input data character
stream, and the current character that was concatenated
25 with the current match in the block 104 "broke" the match.
At this point, a block 106 provides the compressed code
signal representative of the longest match. This code
of the longest match is found in the current match
register 24 and is the node number of the current match.

30 In a block 107 the contents of the current match
register 24 is transferred to the previous match register
25. Thus, the previous match register 25 now stores
the node number of the location representative of the
current longest match. The previous match register 25
35 is then utilized in updating the dictionary 13 in a manner
to be further discussed.

After the current match is stored as the previous

- 18 -

1 match, the current character is stored as the current
match in a block 110. The block 110, therefore, begins
the search for the next longest matched string utilizing,
as the first or root character of the next match, the
5 mismatching input character that broke the last match.
In the block 110, therefore, the current match register
24 is set to the node number of the initialized single
character root node having the value of the current
character. This can be done either algorithmically or
10 by searching in the manner described above with respect
to the block 105. The block 110 leads into the dictionary
updating logic to be described.

The block 110 may alternatively be implemented
with the following logic. Instead of the current match
15 being set to the current character, as in block 110,
the current match can be set to zero and processing can
be returned to the input of the block 104. The result
of this alternative processing always leads to the YES
path of the block 104 because of the dictionary
20 initialization and thus to the block 105. It is
appreciated that the same result is achieved by the block
110, as shown, with fewer processing steps. This logic
is, however, used in the non-initialized compression
processing of Figure 9.

25 When the block 105 is reached, the current
character extension of the current match has been found
in the dictionary and, in accordance with the invention,
the current character is utilized to extend the previous
match to provide an update string to be stored in the
30 dictionary 13. When, however, the block 105 is reached
after only the first input character has been received,
there will be no previous match and the dictionary 13
should not, at that point, be updated. Accordingly,
a decision block 111 determines if the previous match
35 is zero. This is accomplished by examining the previous
match register 25 which was initially set to zero at
block 102. If the previous match is zero, the YES path

1 is taken from the block 111 bypassing the dictionary
updating. When the block 105 is reached after the first
input character has been processed, the previous match
will be non-zero and the NO path from the block 111 will
5 be taken to perform the dictionary updating in accordance
with the present invention.

Accordingly, at a block 112, the code generator
26 provides the next available code. The next available
code will be the node number of the next available empty
10 dictionary location for storing the update string. At
a block 113, the previous match concatenated by the
current character is stored in the dictionary 13 at this
next available empty location accessed by this next
available code.

15 The storage of block 113 is achieved as follows.
The parent pointer field 70 of the next available empty
location accessed by the next available code of block
112 is set to the node number of the previous match found
in the previous match register 25. The character field
20 17 of this next empty location is set to the value of
the current character from the current character register
23. The previous match parent node is linked to this
newly created node as follows. The node number of the
previous match parent node is in the previous match
25 register 25. If the previous match parent node is
childless (child pointer field = 0), the next available
code from the block 112, which is the node number of
the newly created child, is inserted into the child
pointer field 71 of this previous match parent. If the
30 previous match parent already has children, this next
available code node number of the newly created node
is inserted into the sibling list of the children of
the previous match parent. This is done by adjusting
a sibling pointer field 72 of a sibling in the list to
35 accommodate the newly created sibling and to accordingly
insert an appropriate sibling node number into the sibling
pointer field 72 of the newly created sibling.

- 20 -

1 A block 114 is used to update the previous match
register 25 so as to point to the node number of the
previously matched string extended by the current
character as described with respect to the block 113.
5 This is accomplished by inserting the node number of
the newly created child or sibling, as described with
respect to the block 113, into the previous match register
25. This node number is the next available code described
with respect to the block 112 and is provided by the
10 code generator 26.

 After the dictionary updating is performed
pursuant to blocks 112-114, a decision block 115 is
entered to determine if the current input character in
the current character register 23 is the last input
15 character in the input data stream. The block 115 is
also entered from the YES path of block 111 to bypass
dictionary updating as previously discussed. If the
current character is the last character, the YES path
from the block 115 is taken to a block 116 where the
20 code of the current match is output. The compressed
code output provided pursuant to the block 116 is found
in the current match register 24. After outputting the
compressed code pursuant to the block 116, a block 117
is entered ending the processing.

25 If, however, the current character in the current
character register 23 is not the last input character,
the NO branch is taken from the block 115 which returns
to the input of the block 103 by a path 118. Pursuant
to the block 103, the next input character is inserted
30 into the current character register 23 and the data
compression processing of Figure 6 is continued.

 If it is desired to temporarily suspend
processing, the suspension is performed at a hold block
119 in the path 118.

35 It is appreciated from the foregoing, that blocks
103-105 control searching the dictionary 13 for the
longest matched string and that block 106 provides the

1 compressed code output corresponding to the longest match.
The block 110 begins the search for a next longest match
beginning with the character that caused the mismatch
in the previous string matching cycle.

5 The blocks 107 and 112-114 control updating the
dictionary 13 in accordance with the invention. When
the block 104 determines that the current input character
has successfully extended the current match, the blocks
112-114 concatenate this character with the previously
10 matched string that is in the process of being extended.
Thus, dictionary updating is immediate and interleaved
on a character-by-character basis with the string search.

It is appreciated that when the current string
being matched is on the same tree path as the previous
15 string being extended, the property of the present
algorithm of efficiently compressing a repeating character
or character group string is achieved. Such an input
string is compressed in two compressed code signals
irrespective of its length as will be further clarified
20 with respect to Figures 8 and 12.

Referring to Figure 7, with continued reference
to Figures 2 and 3b, a control flow chart is illustrated
depicting the operations executed by the recover and
update control section 50 for decompressing the compressed
25 code generated in accordance with Figure 6. Figure 7
is predicated on the decompression dictionary 43
initialized with all single character strings. Control
57 is considered as containing appropriate circuitry,
such as state machines, to control execution of the
30 operations.

Pursuant to a block 130, the decompression
dictionary 43 is cleared and initialized. The operations
of the block 130 with respect to the dictionary 43 are
the same as those described above with respect to the
35 block 100 and compression dictionary 13.

At a block 131, the previous string register
54 is cleared to zero and at a block 132 an input

- 22 -

1 compressed code is inserted into the current received
code register 53.

Processing continues with a decision block 133
where it is determined if the current received code in
5 the current received code register 53 has a corresponding
string in the dictionary 43. Normally, the dictionary
43 will contain a string corresponding to the current
received code. An exception occurs when the current
received code resulted from the compressor encountering
10 a repeating character or character group string. The
decision of block 133 is effected by accessing the
dictionary 43 using the current received code as an
address and determining whether or not the accessed
dictionary location is cleared. If the dictionary
15 location is cleared, the string corresponding to the
current received code is not in the dictionary.
Alternatively, in the sequential code assignment
embodiment, the decision of block 133 may be effected
by determining if the current received code is less than
20 or equal to the extant code of the code generator 56.
When the current received code is less than or equal
to the extant code of the code generator 56, a string
corresponding to the current received code is in the
dictionary 43. If, however, the current received code
25 is greater than the extant code, the string corresponding
to the current received code is not yet in the dictionary
43.

If the current received code string is not in
the dictionary 43, the NO path from the block 133 is
30 taken to the block 55 for executing unrecognized code
processing. The details of unrecognized code processing
will be described with respect to Figure 8.

When the current received code has a corresponding
string in the dictionary 43, the YES path from the block
35 133 is taken to a block 134. At block 134, the characters
of the string corresponding to the current received code
are recovered by an appropriate known dictionary look-up

- 23 -

1 process (e.g. Welch patent ('302) Figure 5 or Clark patent
('591) Figure 5). A parameter n is provided at a block
135 and is set equal to the number of characters in the
string recovered in block 134. An index i is set equal
5 to 1 at a block 136. The index i is used to step through
the n characters of the string recovered in the block
134 so that the characters of the recovered string are
output beginning with the first character thereof.
Accordingly, a block 137 provides for outputting the
10 i^{th} character of the current received code string.

A decision block 140 is included to determine
if the previous string is equal to zero. This test is
effected by determining if the contents of the previous
string register 54 are zero. If the previous string
15 register 54 is zero, the YES path from the block 140
is taken to bypass dictionary updating. The function
of the block 140 is similar to that described above with
respect to the block 111 of Figure 6 and, therefore,
the YES path from the block 140 is taken only in response
20 to the first received input code.

When the previous string register 54 is not zero,
the NO path from the block 140 is taken and at a block
141 the next available code for the next empty location
of the dictionary 43 is provided by the code generator
25 56. At a block 142 the previous string concatenated
with the i^{th} character of the string corresponding to
the current received code is stored at this next empty
location. At a block 143 the previous string register
54 is updated to store the node number of the extended
30 previous string of block 142. The operations performed
in executing the blocks 141-143 are the same as those
described above with respect to the blocks 112-114 of
Figure 6. In Figure 7 the previous string register 54
is utilized, whereas in Figure 6 the previous match
35 register 25 is involved.

At a block 144, the index i is incremented by
one. The YES path from the decision block 140 is also

- 24 -

1 applied at an input to block 144 so that the dictionary
update processing of blocks 141-143 is bypassed as
previously described. At a decision block 145 a test
is made to determine if the index i has attained the
5 value $n+1$. When the index i is not equal to $n+1$, the
NO path is taken from block 145 which returns processing
to the input of the block 137. Processing through the
blocks 137 and 140-145 is continued until $i=n+1$.

In this manner, the n characters of the current
10 received code string are output in the correct order
at the block 137 and the previous recovered string is
extended by all of the prefixes of the current received
code string. The processing of the blocks 141-143 stores
the same strings in the decompression dictionary 43 as
15 are stored by the blocks 112-114 of Figure 6 in the
compression dictionary 13. The strings stored pursuant
to the blocks 141-143 in the decompression dictionary
43 are stored at the same respective addresses as the
strings stored pursuant to the blocks 112-114 in the
20 compression dictionary 13.

When the index i attains the value $n+1$, the YES
path from the block 145 is taken to a block 146. At
block 146 the current received code string replaces the
previous string in preparation for processing the next
25 input code. This is achieved by inserting the contents
of the current received code register 53 into the previous
string register 54. The unrecognized code processing
55 exits as an input into block 146.

If the current received code just processed is
30 not the last input code, a decision block 147 returns
processing to the input of block 132 via the NO path
of block 147. Processing returns to the block 132 via
a path 148 to initiate processing the next input
compressed code. If it is desired to temporarily suspend
35 processing, the suspension is performed at a hold block
149 in the path 148. The hold block 149 in the
decompressor corresponds to the hold block 119 in the

1 compressor.

If, at the block 147, the current received code is the last input code, the YES path from the block 147 is taken to block 150 to terminate processing.

5 It is appreciated from the foregoing that the blocks 134 and 137 recover and output the characters of the string corresponding to the current received code, while the blocks 141-143 update the dictionary 43 by storing the previous recovered string extended by the
10 prefixes of the current recovered string.

Referring to Figure 8, details of the unrecognized code processing 55 are illustrated. At a block 160, the index i is set equal to one and, for reasons to be described, will be incremented modulo n . At a block
15 161, the n characters of the previous string are recovered. The previous string has n characters since this was the current received code string in the previous string recovery cycle. The block 161 is implemented by accessing the dictionary 43 with the contents of the
20 previous string register 54 utilizing the known dictionary string recovery process as discussed above with respect to the block 134.

At a block 162, the code generator 56 provides the next available code for the next empty dictionary
25 location. At a block 163, the previous string extended by the i^{th} character of the previous string is stored at this next empty location. At a block 164, the previous string is replaced by the extended previous string of block 163 by updating the previous string register 54
30 to store the node number of this extended previous string. The dictionary updating operations of blocks 162-164 are similar to those described above with respect to blocks 141-143. As previously explained, the
35 143 had been described in detail with respect to blocks 112-114 of Figure 6. In the processing of the blocks 162-164, the previous string register 54 is utilized.

- 26 -

1 At a decision block 165, a test is made to
determine if the code currently provided by the code
generator 56 is equal to the current received code in
the current received code register 53. If the extant
5 code of the code generator 56 has not attained the value
of the current received code, the NO path from the block
165 is taken to a block 166 wherein the index i is
incremented modulo n by one. Processing then loops back
to the input of block 162 to store additional update
10 strings until the extant code of the code generator 56
equals the current received code.

When block 165 indicates that the extant code
of the code generator 56 is equal to the current received
code in the current received code register 53, the YES
15 path from the block 165 is taken to a block 167. It
is appreciated that when the block 165 indicates the
code is equal to the current received code, the string
corresponding to this unrecognized current received code
is now stored in the decompression dictionary 43.

20 At the block 167 the characters of the string
corresponding to the current received code are recovered
and each character is output beginning with the first
character. The block 167 is implemented by accessing
the dictionary 43 with the contents of the current
25 received code register 53 and utilizing the known
dictionary string recovery procedures discussed above
with respect to the block 134.

It is appreciated from the foregoing that by
the processing of the blocks 160-167, the string
30 corresponding to the unrecognized compressed input code
is constructed, stored in the decompression dictionary
43 and the characters thereof recovered for outputting.
It is further appreciated that when the compressor of
Figure 6 generates and stores the strings associated
35 with an unrecognized code, the compressor stores n strings
beyond the string corresponding to the transmitted code.
This will be further clarified with respect to Figure 12.

- 27 -

1 The processing at the decompressor to construct and store these n strings is as follows.

Processing proceeds to a block 170 whereat the index i is incremented modulo n by one. Blocks 171-173
5 duplicate the processing of blocks 162-164, respectively. Processing then proceeds to a block 174 whereat the parameter n is decremented by one. The parameter n is then tested in a decision block 175 to determine if n is equal to zero. If n is not yet zero, the NO path
10 is taken from the block 175 back to the input of the block 170. When the parameter n attains the value of zero, the unrecognized code processing exits on the YES path of the block 175.

In the blocks 166 and 170, the index i is
15 incremented modulo n to facilitate providing the appropriate character values for the strings stored pursuant to blocks 163 and 172. The value of n used is that provided by block 135 (Figure 7) and prior to decrementing in block 174. The character values are
20 those of the n characters of the previous string recovered pursuant to block 161 and as indexed by i. The n characters of the previous string recovered pursuant to block 161 form an n character prefix for the strings constructed and stored pursuant to blocks 163 and 172.

25 The processing of Figure 8 functions for any type of code assignment process used by the code generators 26 and 56 including sequential or pseudorandom code assignment such as by hashing. When the code assignment process is sequential, the logic of Figure
30 8 can be simplified as follows.

For sequential code assignment, the test of the block 165 becomes "Code = Current Received Code + n". The blocks 170-175 are eliminated and the unrecognized code processing exits from block 167.

35 It is appreciated from the foregoing that block 167 recovers the characters of the string corresponding to the unrecognized received code and that blocks 163

1 and 172 store the same strings in the decompression dictionary 43 as are stored in the compression dictionary 13 when the repeated character or character group string occurs as discussed above with respect to Figure 6.

5 Referring to Figure 9, with continued reference to Figures 1 and 3b, a control flow chart of detailed operations to be executed by the search and update control section 20 so as to perform data compression in accordance with the invention is illustrated. Control 27 is
10 considered as containing appropriate circuitry, such as state machines, to control execution of the operations. The flow chart of Figure 9 is predicated on a non-initialized compression dictionary 13. In the non-initialized embodiment of Figure 9 when a character is
15 encountered for the first time, a zero code is transmitted followed by transmission of the character in uncompressed form. The zero code provides an indication to the decompressor that such a character has been transmitted by the compressor. The character that is encountered
20 for the first time is stored in the compressor dictionary 13 to function as a stored single character string or root node with respect to subsequent encounters of the character. Except for the accommodation of the characters encountered for the first time, the flow chart of Figure
25 9 functions in the same manner as that of Figure 6.

At a block 180 the dictionary 13 is cleared. Dictionary clearing may be performed by setting fields 17 and 70-72 of Figure 3b to zero.

The flow chart of Figure 9 includes blocks 181-187
30 and 190-194. These blocks are the same as blocks 101, 103-107 and 112-117, respectively, of Figure 6. The explanations given above with respect to these blocks of Figure 6 apply to the corresponding blocks of Figure 9 except as follows.

35 In block 183, when the current match is zero, a search in the dictionary 13 is effected to determine if the single character string has already been entered

1 into the dictionary as a root node. If the single
character string is already in the dictionary, the YES
path from the block 183 is taken, otherwise the NO path
is taken. In block 184, when the current match is zero,
5 the current match register 24 is updated with the node
number of the single character root node that was matched
in the block 183. Additionally, with respect to the
block 187 in a sequential code assignment embodiment,
the code assignments will begin with unity since, in
10 this non-initialized embodiment, all dictionary locations
are available for storing strings as they are encountered
in the input. It is also noted that the block 111 in
Figure 6 that bypasses the dictionary updating is not
required in Figure 9. This follows because, in this
15 non-initialized embodiment, the first input character
is a character that is encountered for the first time
providing a previous match for the next iteration as
will be described below.

Additionally, the block 185 corresponds with
20 the block 106 of Figure 6. In the block 185, as in block
106, the compressed output code is found in the current
match register 24. In the block 185, however, when the
current match is zero, this zero code is output indicating
that the current input character is a character that
25 has been encountered for the first time.

The flow chart of Figure 9 includes a decision
block 195 that determines if the contents of the current
match register 24 is equal to zero. If the current match
is not zero, the NO path from the block 195 is taken
30 to the block 186 whereat the contents of the current
match register 24 is transferred to the previous match
register 25 as described above with respect to
corresponding block 107 of Figure 6. At a block 196,
the current match register 24 is set to zero and
35 processing transfers to the input of the block 183.
The block 186 sets up the dictionary updating for the
next string parsing operation and the block 196 causes

- 30 -

1 the next string search to begin with the character in
the current character register 23.

If the YES path from the block 195 is taken,
the current match is equal to zero and the character
5 in the current character register 23 has been encountered
for the first time. Thus, at a block 197 this character
is output. Since the block 185 had output the zero valued
current match, this first encountered current character,
that was output at the block 197, was preceded by the
10 zero code as discussed above. At a block 200, the next
available code is provided by the code generator 26
indicating the next available empty location in the
dictionary 13. At a block 201, the current character
in the current character register 23 is stored at this
15 next empty location as a single character string root
node. The function of the block 201 is accomplished
by storing the character from the current character
register 23 into the character field 17 of this next
empty location. The parent pointer field 70, child
20 pointer field 71 and sibling pointer field 72 will all
have been previously zeroed at the block 180.

At a block 202, the previous match register 25
is set to the root node created in the block 201 for
the purpose of dictionary updating in the next string
25 parsing operation. This is achieved by inserting into
the previous match register 25, the node number of this
root node newly created at block 201. This node number
will be the code just provided at block 200.

A decision block 203 is included to determine
30 if the character in the current character register 23
is the last input character. If not, the NO path from
the block 203 is taken to the input of the block 182
to acquire the next data character signal from the input
stream. If, however, the current character tested at
35 the block 203 is the last input character, the YES path
from the block 203 is taken to the block 194 to terminate
the processing.

1 It is appreciated from the foregoing that blocks
182-184 control searching the dictionary 13 for the
longest matched string and that block 185 provides the
compressed code output corresponding to the longest match.
5 The block 185 also provides the zero code output that
precedes transmission of a character encountered for
the first time. The block 196 begins the search for
a next longest match by zeroing the current match register
24. Thus, the search for the next longest match begins
10 with the character that caused the mismatch in the
previous string matching cycle, which character is in
the current character register 23.

 The blocks 186, 187, 190 and 191 control updating
the dictionary 13 in accordance with the invention.
15 When the block 183 determines that the current input
character has successfully extended the current match,
the blocks 187, 190 and 191 concatenate this character
with the previously matched string that is in the process
of being extended. The blocks 195, 197 and 200-202
20 control the management of characters that are encountered
for the first time. The block 202 provides such a
character as a previous match for potential extending
in the next string matching cycle. It is appreciated
from the logic of Figure 9 that if several such first
25 encountered characters are received sequentially, only
the last of these characters will be extended in the
next string matching cycle.

 Referring to Figure 10, with continued reference
to Figures 2 and 3b, a control flow chart depicting the
30 operations executed by the recover and update control
section 50 for decompressing the compressed code generated
in accordance with Figure 9 is illustrated. The flow
chart of Figure 10 is predicated on a non-initialized
decompression dictionary. Control 57 is considered as
35 containing appropriate circuitry, such as state machines,
to control execution of the operations.

 At a block 210, the decompression dictionary 43

1 is cleared in the manner described above with respect
to the block 180 of Figure 9. Thus, the decompression
dictionary 43 is cleared in identically the same manner
as the dictionary 13 of the non-initialized embodiment
5 of Figure 9.

The operations performed by the decompression
flow chart of Figure 10 are the same as those performed
by the decompression flow chart of Figure 7 except with
respect to managing the characters encountered for the
10 first time as discussed above with respect to Figure 9.
Accordingly, Figure 10 includes blocks 55, 211-217 and
220-226 which perform the same functions as the blocks
55, 132-137, 141-147 and 150 of Figure 7, respectively.
The descriptions given above with respect to blocks 55,
15 132-137, 141-147 and 150 of Figure 7 apply to the
corresponding blocks of Figure 10.

A block corresponding to the block 140 of Figure
7 is not utilized in Figure 10. As previously described,
the block 140 bypasses dictionary updating when the
20 previous string register 54 is zero. This occurs in
response to the first received input code in the
initialized embodiment of Figure 7. In the
non-initialized embodiment of Figure 10, the first
received input code is for a character encountered for
25 the first time, thereby providing a previous string for
updating in subsequent cycles as will be described.

The processing of the blocks 217, 220 and 221
stores the same strings in the decompression dictionary
43 as are stored by the blocks 187, 190 and 191 of Figure
30 9 in the compression dictionary 13. The strings stored
pursuant to the blocks 217, 220 and 221 in the
decompression dictionary 43 are stored at the same
respective addresses as the strings stored pursuant to
the blocks 187, 190 and 191 in the compression dictionary
35 13. Furthermore, when the unrecognized code processing
of block 55 (Figure 8) is performed in the context of
Figure 10, the blocks 163 and 172 of Figure 8 store the

- 33 -

1 same strings in the decompression dictionary 43 as are
stored in the compression dictionary 13 when the repeated
character or character group string occurs with respect
to the operation of the compressor of Figure 9.

5 Specifically, the block 212 corresponds with
the block 133 of Figure 7. The descriptions given above
with respect to the block 133 apply to the block 212
except with respect to the zero received code that
precedes receipt of a character that was encountered
10 for the first time by the compressor. It is appreciated,
however, that when processing reaches the block 212,
the current received code will not be zero since this
situation is handled in another branch of Figure 10 now
to be described.

15 At the block 211, an input compressed code signal
is inserted into the current received code register 53.
A decision block 230 tests the contents of the current
received code register 53 to determine if the current
received code is zero. If the current received code
20 is not zero, the NO path from the block 230 is taken
as an input to the block 212 where processing proceeds
as previously described.

If, however, the current received code is zero,
a character that was encountered for the first time by
25 the compressor of Figure 9 is expected. Accordingly,
this character is input at a block 231. The current
received code register 53 may be utilized to temporarily
hold this character. At a block 232, the code generator
56 provides the next available code corresponding to
30 the next empty location of the decompression dictionary
43. At a block 233 the input character is stored at
this next empty location as a single character root node.
This is accomplished at block 233 in the manner described
above with respect to the block 201 of Figure 9. Thus,
35 the decompression subsystem 40 stores the same single
character strings in the decompression dictionary 43,
and at the same addresses, as the compression system 10

- 34 -

1 stores in the compression dictionary 13 in the
non-initialized embodiment of Figure 9.

At a block 234, the character received at the
block 231 is output to maintain consistency between the
5 recovered data output of the decompressor and the input
data received at the compressor. This may be accomplished
by outputting this character from the current received
code register 53 where the character was temporarily
stored.

10 At a block 235, the parameter n is set to one.
If this character, which was encountered for the first
time and just processed, should repeat at the input of
the compressor, thereby generating an unrecognized
compressed code, n=1 is the appropriate value for the
15 unrecognized code processing 55 that would be performed
in the next decompressor cycle.

At a block 236, the previous string register
54 is set to the root node created in the block 233 for
the purpose of dictionary updating in the next string
20 recovery cycle. This is achieved by inserting the node
number of this root node newly created at the block 233
into the previous string register 54. This node number
will be the code just provided at block 232. It will
be appreciated from the logic of Figure 10 that if several
25 characters, each encountered for the first time, are
received sequentially, only the last such character will
be extended in the next string recovery cycle in a manner
similar to that described with respect to Figure 9.

The output of the block 236 provides an input
30 to the block 225 to continue the processing as described
above.

Referring to Figures 11a-11e, consecutive states
of the compression dictionary 13, when compressing a
typical input data stream, are illustrated. The
35 dictionary states are schematically represented by partial
searchtrees. As indicated, the input stream being
compressed is "abcfghx". The commas illustrated,

- 35 -

1 representing string parsing, are virtual and not in the
input stream. Underscoring indicates the current input
character. As indicated in Figure 11a, the dictionary
13 initially stores the strings "abc" and "fgh". In
5 Figure 11a the string "abc" has just been matched as
the longest matched string and the code of "abc" is output
as indicated by an arrow 240. The string "abc" has,
therefore, been parsed from the input as indicated by
a virtual comma 241.

10 In Figure 11b, the next input character "f"
matches the first character of the string "fgh" as
indicated by an arrow 242. In accordance with the
invention, the string "abc", transmitted as the previous
compressed code in Figure 11a, is extended by the
15 character "f" as indicated by an arrow 243.

In Figure 11c, the next input character "g"
matches the second character of the stored string "fgh"
and the previous extended string "abcf" is now extended
by the matching input character "g". Similarly, in Figure
20 11d the input character "h" matches the third character
of the string "fgh" and is appended to the growing
previous string to now form the string "abcfgh".

In Figure 11e, the next input character "x" breaks
the match of the string "fgh" and the code of the longest
25 match "fgh" is output as indicated by an arrow 244.
The string "fgh" has been parsed from the input as
indicated by a virtual comma 245.

It is appreciated from Figures 11a-11e that the
stored string "abc" has been extended by all of the
30 prefixes of the matched string "fgh" and that the
dictionary updating is immediate and interleaved with
the matching of each of the characters of the current
matched string. Thus, in Figure 11b the string "abcf"
was available for matching in the next iteration. In
35 Figure 11c the string "abcfg" was available for matching
in the next iteration and in Figure 11d the string
"abcfgh" was available for matching in the next iteration.

1 It is further appreciated from Figures 11a-11e
that the decompressor receives the consecutive compressed
output codes for "abc" and "fgh". When the output code
for "fgh" is received, the decompressor utilizes the
5 recovered string "fgh" and the string "abc", corresponding
to the previously received compressed code, to construct
and store the strings described and illustrated in Figures
11a-11e.

Referring to Figures 12a-12g, the manner in which
10 the immediate and interleaved dictionary updating aspect
of the present invention provides a run length encoding
advantage is demonstrated. As discussed above, the
present invention compresses a repeating character or
character group run in two compressed code symbols
15 regardless of the length of the run. Figures 12a-12g
are schematic representations of partial searchtrees
illustrating consecutive states of the compression
dictionary 13 when the input data character stream is
a repeating character group. The input data stream is
20 illustrated as "abababax". In this sample input it is
noted that the repeating character group run terminates
with a fragment of the character group. The operations
described below are also applicable to termination of
the run by the complete character group. As in Figures
25 11a-11e, the virtual commas represent string parsing
and the underscoring represents the current input
character.

In Figure 12a, the compression dictionary 13
is storing the string "ab" which is matched by the first
30 two input characters. Thus, the code of "ab" is output
as indicated by an arrow 250. The parsing of the string
"ab" from the input is indicated by a virtual comma 251.

In Figure 12b, the current input character "a"
matches the first character of the stored string "ab",
35 as indicated by an arrow 252. Since the code for "ab"
was transmitted as the previous compressed code, the
matching character "a" is appended to the string "ab"

1 as indicated by an arrow 253. The compression dictionary
13 now stores the string "aba" which can be utilized
for matching.

In Figure 12c, the next input character "b"
5 matches the second character of the stored string "aba"
and, accordingly, the string "aba" is extended by this
character. Thus, the compression dictionary 13 now stores
the string "abab" which is available for matching.

This sequence of input character matching and
10 string extending is illustrated for the next three input
characters in Figures 12d-12f. Thus, in Figure 12d the
string "ababa" is stored and available for matching,
in Figure 12e the string "ababab" is stored and available
for matching, and in Figure 12f the string "abababa"
15 is stored and available for matching.

It is appreciated that as this repeating sequence
continues, the input character matching and stored string
extending is performed on the same branch of the tree
and would continue until a character is received that
20 would break the match. As this is occurring, the code
generator 26 (Figure 1) is continuously providing new
codes so as to store the strings at the newly created
nodes. Up to this point, the decompressor is unaware
of the activities, as illustrated in Figures 12b-12f,
25 occurring at the compressor. The last information
received by the decompressor was the output code of the
string "ab" as described with respect to Figure 12a.

In Figure 12g, the input character "x" breaks
the match since the string "ababax" is not found in the
30 compressor dictionary 13. Accordingly, the code of the
longest match "ababa" is output as indicated by an arrow
254. This string has, accordingly, been parsed from
the input as indicated by a virtual comma 255. It is
appreciated from Figure 12g that two additional strings
35 have been stored in the compressor dictionary 13 beyond
the longest matched string. These strings are represented
by nodes 256 and 257.

1 Thus, it is appreciated from Figures 12a-12g
that the repeating character group string "ababa" was
compressed in two code symbols as indicated by reference
numerals 250 and 254. It is further appreciated that
5 if the repeating run of the code group "ab" had been
continued beyond Figure 12f for a longer run than
illustrated, only two code symbols would still be utilized
for the compression. It is seen, that the repeating
run ends at a fragment of the repeating character group
10 but could also continue to terminate at the complete
group "ab".

It is appreciated from the foregoing that the
output code indicated in Figure 12g by reference numeral
254 will be unrecognized at the decompressor since the
15 decompressor is not yet storing the string "ababa".
The unrecognized code processing of Figure 8 constructs
and stores the string "ababa", as well as the prefixes
thereof shown in Figures 12b and 12c. The unrecognized
code processing furthermore constructs and stores the
20 additional extended strings designated by reference
numerals 256 and 257 in Figure 12g.

In Figure 8 the processing of blocks 160-167
constructs, stores and outputs the string "ababa"
corresponding to the unrecognized compressed code
25 indicated by the arrow 254. The processing of blocks
170-175 constructs and stores the additional strings
256 and 257.

In the example of Figures 12a-12g, the previous
string used in the unrecognized code processing is the
30 string "ab" which was already stored in the decompressor
dictionary when the decompressor performed the string
recovery cycle corresponding to Figure 12a. In Figure
8, the parameter n is 2 and the index i is incremented
modulo 2 thereby sequentially and repeatedly providing
35 the characters "ab" for the blocks 163 and 172 so as
to construct the strings illustrated in Figure 12b-12f.

It is further appreciated from the foregoing

1 that the unrecognized code processing constructs and
recovers the appropriate strings when the repeating
character or repeating character group run is terminated
by a mismatching character or by running out of input
5 characters. The run may terminate with a fragment of
the multi-character group of a repeating character group
run or with the complete group. Figures 12a-12g
illustrate termination by the mismatching character "x"
and further illustrate termination with a fragment of
10 the repeating group. In Figure 12g the run ends with
the fragment "a" of the repeating group "ab". When the
repeating group run terminates with the complete group
"ab", the appropriate operations are readily appreciated
from the above-given descriptions.

15 It is appreciated from the foregoing that the
above-described processing preserves the prefix property
for the strings stored in the dictionaries 13 and 43
in that the prefixes of the stored strings also exist
in the dictionaries.

20 The above embodiments compress a stream of input
data character signals. The input data characters can
be over any size alphabet having any corresponding
character bit size. For example, the data characters
can be textual, such as ASCII characters, over an
25 alphabet, such as the 256 character ASCII alphabet of
eight-bit characters. The input data can also be binary
characters over the two character binary alphabet 1 and
0 having a one-bit sized character. This type of alphabet
occurs, for example, in bit mapped images. It is
30 appreciated that textual data can also be compressed
over the two character alphabet of the underlying binary
data.

The above embodiments were described in terms
of searching for the longest match. It is appreciated
35 that the immediate and interleaved updating process of
the present invention can also be used in systems that
match strings that are not necessarily the longest.

1 It is appreciated that the above-described
embodiments of the invention may be implemented in
hardware, firmware, software or a combination thereof.
Discrete circuit components may readily be implemented
5 for performing the various described functions. In a
software embodiment, appropriate microprocessors,
programmed with coding readily generated from the
above-described flow charts, may be utilized.

 While the invention has been described in its
10 preferred embodiments, it is to be understood that the
words which have been used are words of description rather
than of limitation and that changes may be made within
the purview of the appended claims without departing
from the true scope and spirit of the invention in its
15 broader aspects.

20

25

30

35

- 41 -

CLAIMS

- 1 1. Data compression apparatus for compressing a
stream of data character signals into a stream of
compressed code signals, comprising
storage means for storing strings of data
5 character signals, each said string having a code signal
associated therewith,
means for searching said stream of data character
signals by comparing said stream to said stored strings
to perform a character-by-character match therewith until
10 a predetermined match is determined,
means for providing the code signal associated
with said predetermined match so as to provide said stream
of compressed code signals,
means for entering into said storage means,
15 interleaved with said character-by-character match,
extended strings comprising a previously matched string
corresponding to a last provided code signal extended
by each data character signal, in turn, as each data
character signal is matched, and
20 means for assigning respective code signals to
said extended strings.
2. The apparatus of Claim 1 wherein said means for
entering is operative to enter one of said extended
25 strings into said storage means as each said data
character signal is matched and before a next data
character signal is matched.
3. The apparatus of Claim 2 wherein said
30 predetermined match comprises the longest match between
said stream of data character signals and said stored
strings,
said means for providing being operative for
providing the code signal associated with said longest
35 match so as to provide said stream of compressed code
signals.

1 4. The apparatus of Claim 3 wherein said apparatus
operates in successive string matching cycles, respective
longest matched strings being determined in said
successive cycles, a current cycle following a previous
5 cycle,

said previously matched string being matched
in said previous cycle with said last provided code signal
being provided in said previous cycle,

said character-by-character match occurring in
10 said current cycle with said previously matched string
being extended by said each data character signal during
said current cycle.

5. The apparatus of Claim 4 wherein said means for
15 searching and said means for entering are operative so
that when a partial string W and a data character signal
C are matched, one of said extended strings is entered
into said storage means with said data character signal
C as an extension character of a string PW where P is
20 said previously matched string and W is in the process
of being matched in said current cycle.

6. The apparatus of Claim 4 wherein said means for
searching is operative for determining when said longest
25 match has been achieved by determining when a data
character signal fails to match during said
character-by-character match,

said means for searching includes means for
beginning a next string matching cycle with said data
30 character signal that failed to match.

7. The apparatus of Claim 1 further including means
for initializing said storage means with all single
character strings with respective codes associated
35 therewith.

- 43 -

1 8. The apparatus of Claim 1 further including
means for outputting a data character signal
encountered for the first time in uncompressed form
following an indication that such a data character signal
5 is being output, and
means for entering said data character signal
encountered for the first time into said storage means
as a single character string.

10 9. The apparatus of Claim 8 wherein said indication
comprises a zero code.

10. The apparatus of Claim 4 wherein said strings
are stored in said storage means in a linked tree
15 structure and said stream of data character signals
includes a repeating character string comprised of a
repeating character,
said apparatus being operative for compressing
said repeating character string in two compressed code
20 signals irrespective of the length thereof.

11. The apparatus of Claim 10 wherein
said previously matched string comprises said
repeating character,
25 said means for searching is operative to match
said repeating character string on a path through said
tree, and
said means for entering is operative to enter
said extended strings on said path,
30 thereby compressing said repeating character
string in two compressed code signals irrespective of
the length thereof.

- 44 -

1 12. The apparatus of Claim 4 wherein said strings
are stored in said storage means in a linked tree
structure and said stream of data character signals
includes a repeating character group string comprised
5 of a repeating character group,
said apparatus being operative for compressing
said repeating character group string in two compressed
code signals irrespective of the length thereof.

10 13. The apparatus of Claim 12 wherein
said previously matched string comprises said
repeating character group,
said means for searching is operative to match
said repeating character group string on a path through
15 said tree, and
said means for entering is operative to enter
said extended strings on said path,
thereby compressing said repeating character
group string in two compressed code signals irrespective
20 of the length thereof.

25

30

35

- 1 14. Data decompression apparatus for receiving a
stream of compressed code signals provided by a data
compressor responsive to a stream of data character
signals, said apparatus operative for recovering said
5 stream of data character signals from said stream of
compressed code signals, comprising
storage means for storing strings of data
character signals, each said string having a code signal
associated therewith,
10 means for accessing said storage means with a
current received compressed code signal for recovering
a string from said storage means corresponding to said
current received compressed code signal so as to recover
the data character signals thereof,
15 means for entering into said storage means,
extended strings comprising a previously recovered string
corresponding to a previously received compressed code
signal extended by each data character signal, in turn,
of said recovered string corresponding to said current
20 received compressed code signal,
means for assigning respective code signals to
said extended strings,
further means for entering into said storage
means, further extended strings in response to receiving
25 an unrecognized compressed code signal, said further
extended strings comprising said previously recovered
string corresponding to said previously received
compressed code signal extended by each data character
signal, in turn, of said previously recovered string,
30 sequentially and repeatedly, until a string corresponding
to said unrecognized compressed code signal is entered,
means for assigning respective code signals to
said further extended strings, and
means for providing the data character signals
35 of said string corresponding to said unrecognized
compressed code signal so as to recover said data
character signals.

1 15. The apparatus of Claim 14 wherein said further
means for entering is operative for entering into said
storage means, additional extended strings comprising
said string corresponding to said unrecognized compressed
5 code signal further extended by the data character signals
of said previously recovered string, in turn, until a
number of said additional extended strings are entered
into said storage means, said number being equal to the
number of data character signals comprising said
10 previously recovered string,

means for assigning respective code signals to
said additional extended strings.

16. The apparatus of Claim 15 wherein said apparatus
15 operates in successive string recovery cycles, respective
strings being recovered in said successive cycles, a
current cycle following a previous cycle,

said previously recovered string being recovered
in said previous cycle with said previously received
20 compressed code signal being received in said previous
cycle,

said current received compressed code signal
being received during said current cycle, and

said means for entering being operative for
25 entering said extended strings into said storage means
during said current cycle.

17. The apparatus of Claim 16 wherein said
unrecognized compressed code signal is received during
30 said current cycle,

said further means for entering being operative
for entering said further extended strings and said
additional extended strings into said storage means during
said current cycle.

35

1 18. The apparatus of Claim 14 further including means
for initializing said storage means with all single
character strings with respective codes associated
therewith.

5

19. The apparatus of Claim 14 wherein said stream
of compressed code signals includes a data character
signal in uncompressed form following an indication that
such a data character signal is being received,

10 said data character signal in uncompressed form
corresponding to a data character signal encountered
for the first time by said data compressor in said stream
of data character signals,

said apparatus further including,

15 means for entering said data character signal
received in uncompressed form into said storage means
as a single character string, and

means for outputting said data character signal
received in uncompressed form.

20

20. The apparatus of Claim 19 wherein said indication
comprises a zero code.

21. The apparatus of Claim 17 wherein

25 said stream of data character signals includes
a repeating character string comprised of a repeating
character, said data compressor being operative for
compressing said repeating character string into two
consecutive compressed code signals, and

30 said strings are stored in said storage means
in a linked tree structure.

35

- 1 22. The apparatus of Claim 21 wherein
said previously received compressed code signal
comprises a first of said consecutive compressed code
signals and said unrecognized compressed code signal
5 comprises a second of said consecutive compressed code
signals,
said previously recovered string comprises said
repeating character and said string corresponding to
said unrecognized compressed code signal comprises said
10 repeating character string,
said previously recovered string is stored on
a path through said tree, and
said means for entering and said further means
for entering are operative for entering said further
15 extended strings and said additional extended strings
on said path.
23. The apparatus of Claim 17 wherein
said stream of data character signals includes
20 a repeating character group string comprised of a
repeating character group, said data compressor being
operative for compressing said repeating character group
string into two consecutive compressed code signals,
and
25 said strings are stored in said storage means
in a linked tree structure.

30

35

1 24. The apparatus of Claim 23 wherein
said previously received compressed code signal
comprises a first of said consecutive compressed code
signals and said unrecognized compressed code signal
5 comprises a second of said consecutive compressed code
signals,
said previously recovered string comprises said
repeating character group and said string corresponding
to said unrecognized compressed code signal comprises
10 said repeating character group string,
said previously recovered string is stored on
a path through said tree, and
said means for entering and said further means
for entering are operative for entering said further
15 extended strings and said additional extended strings
on said path.

20

25

30

35

- 50 -

- 1 25. Data decompression apparatus for recovering a
stream of data character signals from a stream of
compressed code signals, comprising
means for receiving said stream of compressed
5 code signals,
storage means for storing strings of data
character signals, each said string having a code signal
associated therewith,
means for accessing said storage means with a
10 current received compressed code signal for recovering
a string from said storage means corresponding to said
current received compressed code signal so as to recover
the data character signals thereof,
means for entering into said storage means,
15 extended strings comprising a previously recovered string
corresponding to a previously received compressed code
signal extended by each data character signal, in turn,
of said recovered string corresponding to said current
received compressed code signal,
20 further means for entering into said storage
means, further extended strings in response to receiving
an unrecognized compressed code signal,
means for assigning respective code signals to
said extended strings and to said further extended
25 strings,
said further means for entering being operative
for generating said further extended strings in accordance
with said previously recovered string, the number of
data character signals comprising said previously
30 recovered string, said unrecognized compressed code signal
and said code signals being assigned to said further
extended strings, one of said further extended strings
being a string corresponding to said unrecognized
compressed code signal, and
35 means for providing the data character signals
of said string corresponding to said unrecognized
compressed code signal so as to recover said data

- 51 -

1 character signals.

26. A data compression method for compressing a stream
of data character signals into a stream of compressed
5 code signals, comprising

storing strings of data character signals in
storage means, each said string having a code signal
associated therewith,

searching said stream of data character signals
10 by comparing said stream to said stored strings to perform
a character-by-character match therewith until a
predetermined match is determined,

providing the code signal associated with said
predetermined match so as to provide said stream of
15 compressed code signals,

entering into said storage means, interleaved
with said character-by-character match, extended strings
comprising a previously matched string corresponding
to a last provided code signal extended by each data
20 character signal, in turn, as each data character signal
is matched, and

assigning respective code signals to said extended
strings.

25 27. The method of Claim 26 wherein said entering
step includes entering one of said extended strings into
said storage means as each said data character signal
is matched and before a next data character signal is
matched.

30 28. The method of Claim 27 wherein said predetermined
match comprises the longest match between said stream
of data character signals and said stored strings,
said providing step comprises providing the code
35 signal associated with said longest match so as to provide
said stream of compressed code signals.

1 29. The method of Claim 28 wherein said method
operates in successive string matching cycles, respective
longest matched strings being determined in said
successive cycles, a current cycle following a previous
5 cycle,

said previously matched string being matched
in said previous cycle with said last provided code signal
being provided in said previous cycle,

10 said character-by-character match occurring in
said current cycle with said previously matched string
being extended by said each data character signal during
said current cycle.

30. The method of Claim 29 wherein said searching
15 and said entering steps are performed so that when a
partial string W and a data character signal C are
matched, one of said extended strings is entered into
said storage means with said data character signal C
as an extension character of a string PW where P is said
20 previously matched string and W is in the process of
being matched in said current cycle.

31. The method of Claim 29 wherein said searching
step includes determining when said longest match has
25 been achieved by determining when a data character signal
fails to match during said character-by-character match,
said searching step including beginning a next
string matching cycle with said data character signal
that failed to match.

30
32. The method of Claim 26 further including
initializing said storage means with all single character
strings with respective codes associated therewith.

- 53 -

1 33. The method of Claim 26 further including
outputting a data character signal encountered
for the first time in uncompressed form following an
indication that such a data character signal is being
5 output, and
entering said data character signal encountered
for the first time into said storage means as a single
character string.

10 34. The method of Claim 33 wherein said indication
comprises a zero code.

35. The method of Claim 29 wherein said storing step
comprises storing said strings in said storage means
15 in a linked tree structure,
said stream of data character signals including
a repeating character string comprised of a repeating
character,
said method compressing said repeating character
20 string in two compressed code signals irrespective of
the length thereof.

36. The method of Claim 35 wherein
said previously matched string comprises said
25 repeating character,
said searching step includes matching said
repeating character string on a path through said tree,
and
said entering step includes entering said
30 extended strings on said path,
thereby compressing said repeating character
string in two compressed code signals irrespective of
the length thereof.

35

- 1 37. The method of Claim 29 wherein said storing step
comprises storing said strings in said storage means
in a linked tree structure,
said stream of data character signals including
5 a repeating character group string comprised of a
repeating character group,
said method compressing said repeating character
group string in two compressed code signals irrespective
of the length thereof.
- 10 38. The method of Claim 37 wherein
said previously matched string comprises said
repeating character group,
said searching step includes matching said
15 repeating character group string on a path through said
tree, and
said entering step includes entering said
extended strings on said path,
thereby compressing said repeating character
20 group string in two compressed code signals irrespective
of the length thereof.
- 25
- 30
- 35

- 55 -

1 39. A data decompression method for receiving a stream
of compressed code signals provided by a data compressor
responsive to a stream of data character signals, said
method recovering said stream of data character signals
5 from said stream of compressed code signals, comprising
storing strings of data character signals in
storage means, each said string having a code signal
associated therewith,
accessing said storage means with a current
10 received compressed code signal for recovering a string
from said storage means corresponding to said current
received compressed code signal so as to recover the
data character signals thereof,
entering into said storage means, extended strings
15 comprising a previously recovered string corresponding
to a previously received compressed code signal extended
by each data character signal, in turn, of said recovered
string corresponding to said current received compressed
code signal,
20 assigning respective code signals to said extended
strings,
further entering into said storage means, further
extended strings in response to receiving an unrecognized
compressed code signal, said further extended strings
25 comprising said previously recovered string corresponding
to said previously received compressed code signal
extended by each data character signal, in turn, of said
previously recovered string, sequentially and repeatedly,
until a string corresponding to said unrecognized
30 compressed code signal is entered,
assigning respective code signals to said further
extended strings, and
providing the data character signals of said
string corresponding to said unrecognized compressed
35 code signal so as to recover said data character signals.

- 1 40. The method of Claim 39 wherein said further
entering step includes entering into said storage means,
additional extended strings comprising said string
corresponding to said unrecognized compressed code signal
5 further extended by the data character signals of said
previously recovered string, in turn, until a number
of said additional extended strings are entered into
said storage means, said number being equal to the number
of data character signals comprising said previously
10 recovered string,
assigning respective code signals to said
additional extended strings.
41. The method of Claim 40 wherein said method
15 operates in successive string recovery cycles, respective
strings being recovered in said successive cycles, a
current cycle following a previous cycle,
said previously recovered string being recovered
in said previous cycle with said previously received
20 compressed code signal being received in said previous
cycle,
said current received compressed code signal
being received during said current cycle, and
said entering step comprises entering said
25 extended strings into said storage means during said
current cycle.
42. The method of Claim 41 wherein said unrecognized
compressed code signal is received during said current
30 cycle,
said further entering step including entering
said further extended strings and said additional extended
strings into said storage means during said current cycle.
- 35 43. The method of Claim 39 further including
initializing said storage means with all single character
strings with respective codes associated therewith.

- 57 -

- 1 44. The method of Claim 39 wherein said stream of
compressed code signals includes a data character signal
in uncompressed form following an indication that such
a data character signal is being received,
5 said data character signal in uncompressed form
corresponding to a data character signal encountered
for the first time by said data compressor in said stream
of data character signals,
said method further including,
10 entering said data character signal received
in uncompressed form into said storage means as a single
character string, and
outputting said data character signal received
in uncompressed form.
15
45. The method of Claim 44 wherein said indication
comprises a zero code.
46. The method of Claim 42 wherein
20 said stream of data character signals includes
a repeating character string comprised of a repeating
character, said data compressor compressing said repeating
character string into two consecutive compressed code
signals, and
25 said storing step comprises storing said strings
in said storage means in a linked tree structure.

30

35

- 58 -

1 47. The method of Claim 46 wherein
said previously received compressed code signal
comprises a first of said consecutive compressed code
signals and said unrecognized compressed code signal
5 comprises a second of said consecutive compressed code
signals,

said previously recovered string comprises said
repeating character and said string corresponding to
said unrecognized compressed code signal comprises said
10 repeating character string,

said previously recovered string is stored on
a path through said tree, and

said entering and said further entering steps
include entering said further extended strings and said
15 additional extended strings on said path.

48. The method of Claim 42 wherein

said stream of data character signals includes
a repeating character group string comprised of a
20 repeating character group, said data compressor
compressing said repeating character group string into
two consecutive compressed code signals, and

said storing step comprises storing said strings
in said storage means in a linked tree structure.

25

30

35

- 59 -

1 49. The method of Claim 48 wherein
said previously received compressed code signal
comprises a first of said consecutive compressed code
signals and said unrecognized compressed code signal
5 comprises a second of said consecutive compressed code
signals,
said previously recovered string comprises said
repeating character group and said string corresponding
to said unrecognized compressed code signal comprises
10 said repeating character group string,
said previously recovered string is stored on
a path through said tree, and
said entering and said further entering steps
include entering said further extended strings and said
15 additional extended strings on said path.

20

25

30

35

- 60 -

- 1 50. A data decompression method for recovering a
stream of data character signals from a stream of
compressed code signals, comprising
- 5 receiving said stream of compressed code signals,
storing strings of data character signals in
storage means, each said string having a code signal
associated therewith,
- accessing said storage means with a current
received compressed code signal for recovering a string
10 from said storage means corresponding to said current
received compressed code signal so as to recover the
data character signals thereof,
- entering into said storage means, extended strings
comprising a previously recovered string corresponding
15 to a previously received compressed code signal extended
by each data character signal, in turn, of said recovered
string corresponding to said current received compressed
code signal,
- further entering into said storage means, further
20 extended strings in response to receiving an unrecognized
compressed code signal,
- assigning respective code signals to said extended
strings and to said further extended strings,
- said further entering step includes generating
25 said further extended strings in accordance with said
previously recovered string, the number of data character
signals comprising said previously recovered string,
said unrecognized compressed code signal and said code
signals being assigned to said further extended strings,
- 30 one of said further extended strings being a string
corresponding to said unrecognized compressed code signal,
and
- providing the data character signals of said
string corresponding to said unrecognized compressed
35 code signal so as to recover said data character signals.

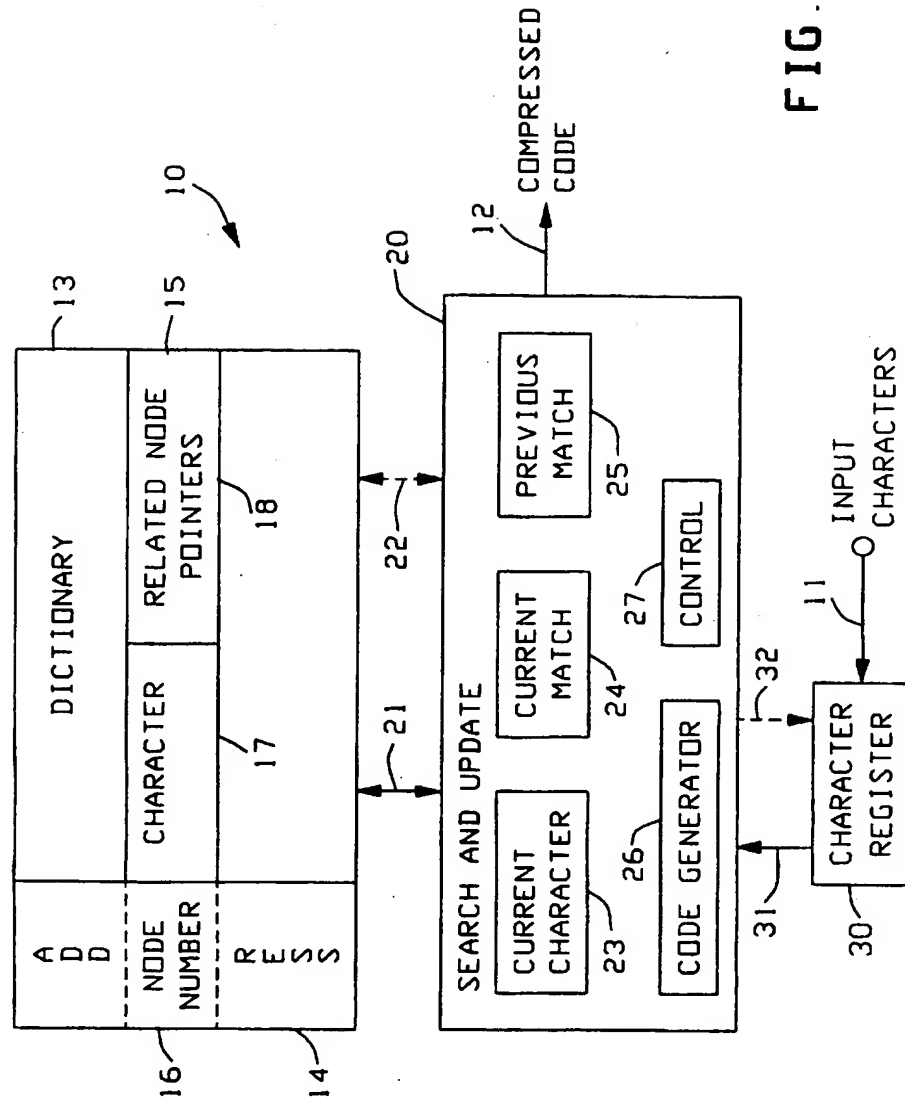


FIG. 1

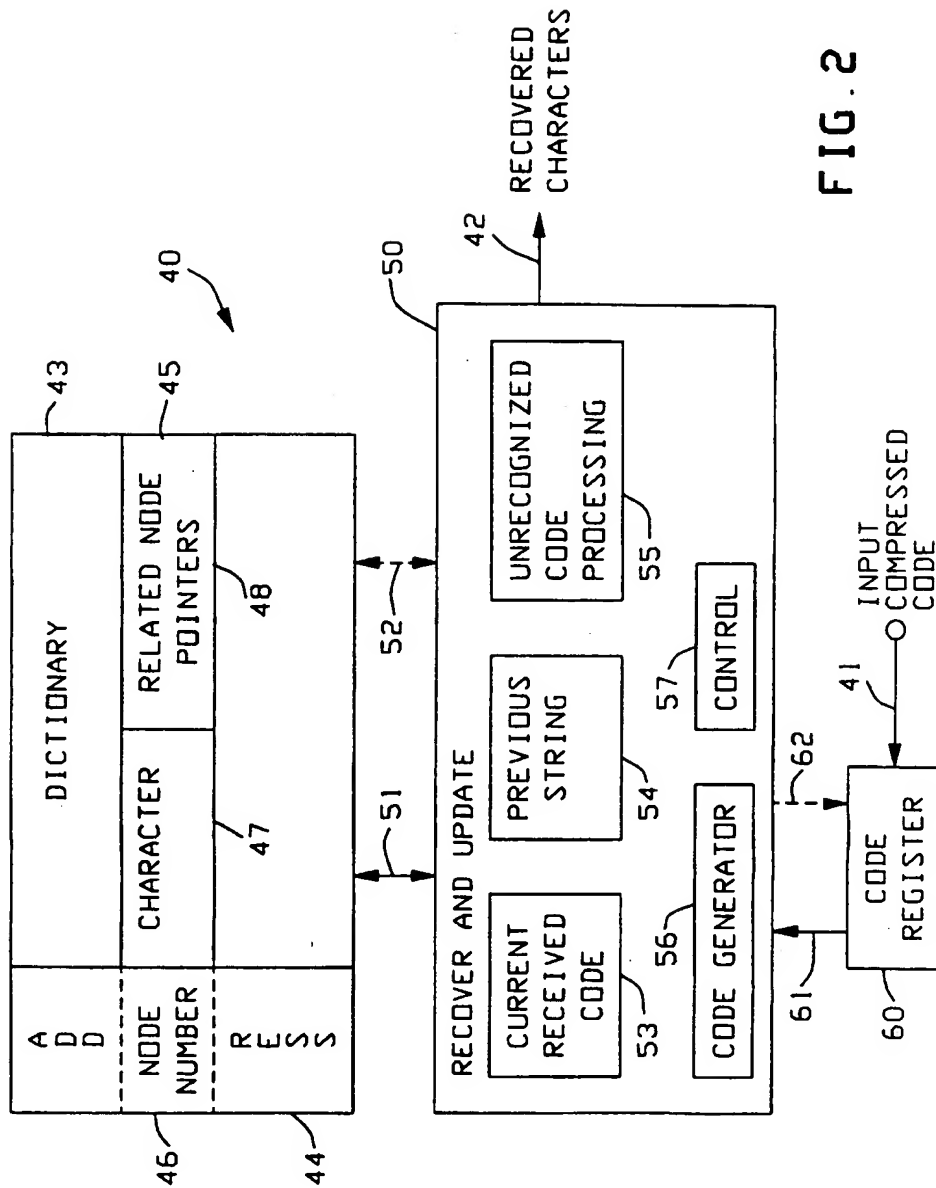


FIG. 2

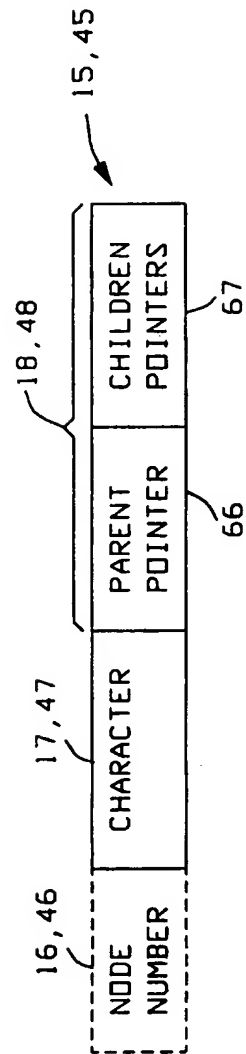


FIG. 30a

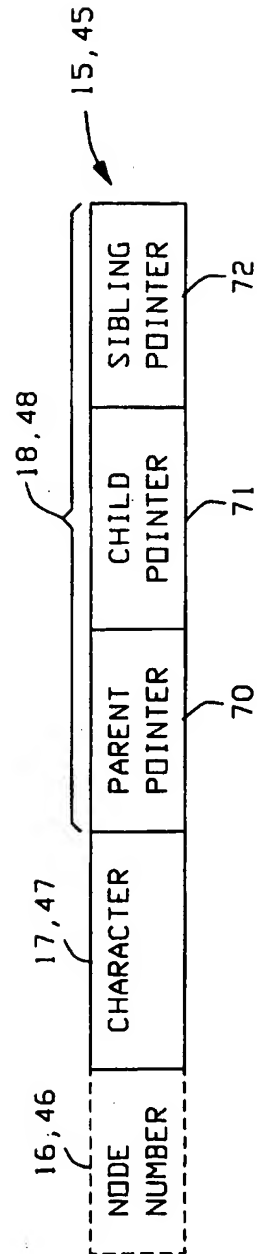


FIG. 30b

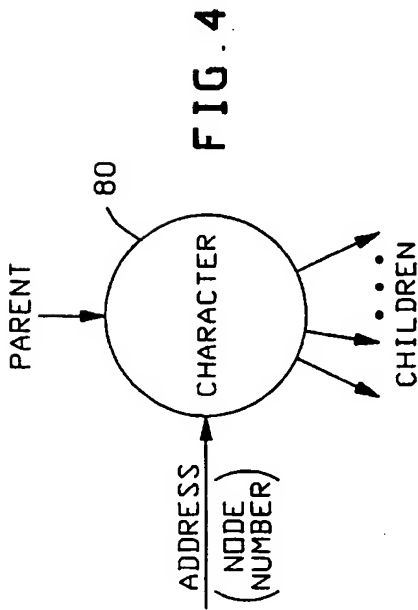


FIG. 4

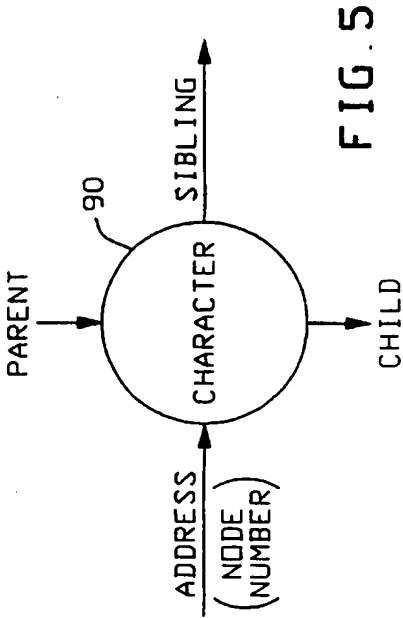


FIG. 5

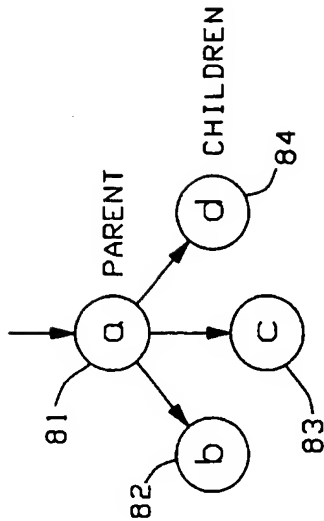


FIG. 40

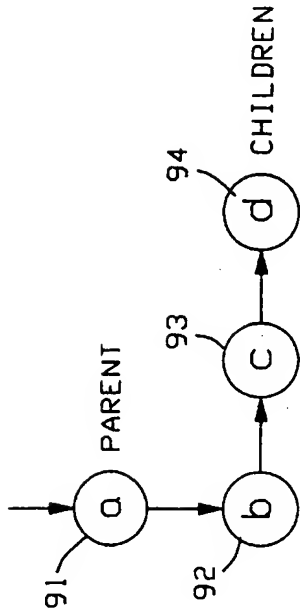
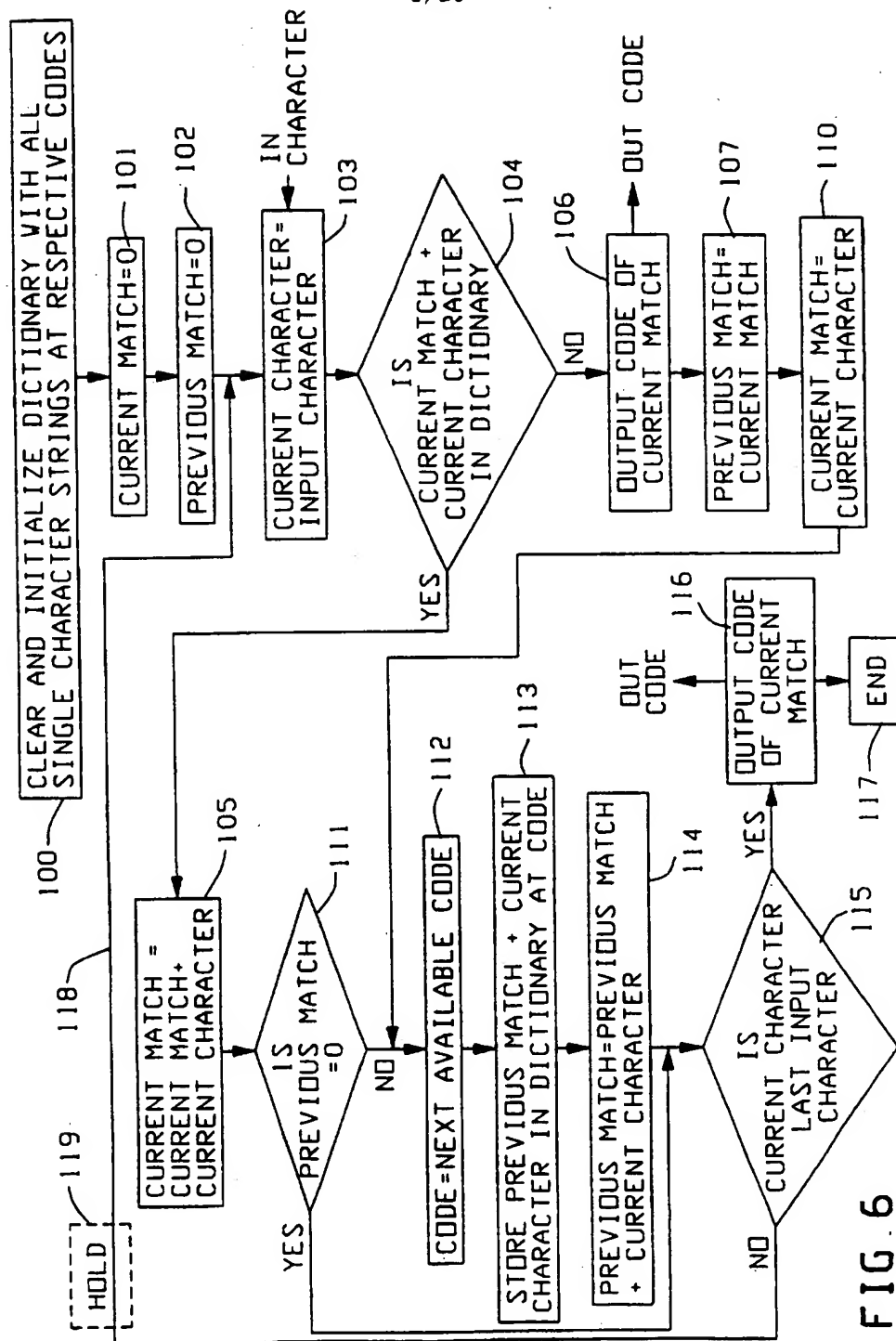


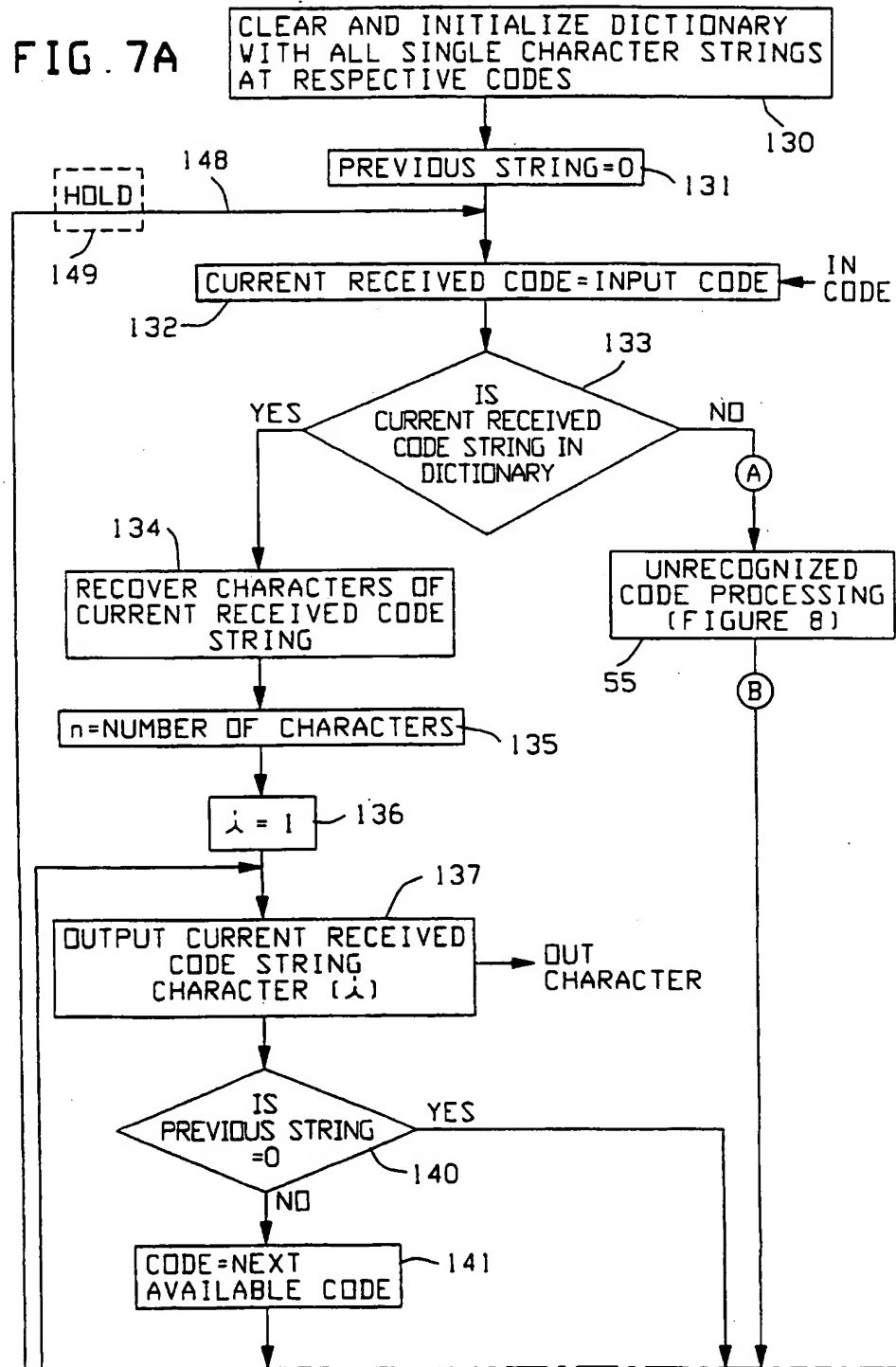
FIG. 50

5/16



6/16

FIG. 7A



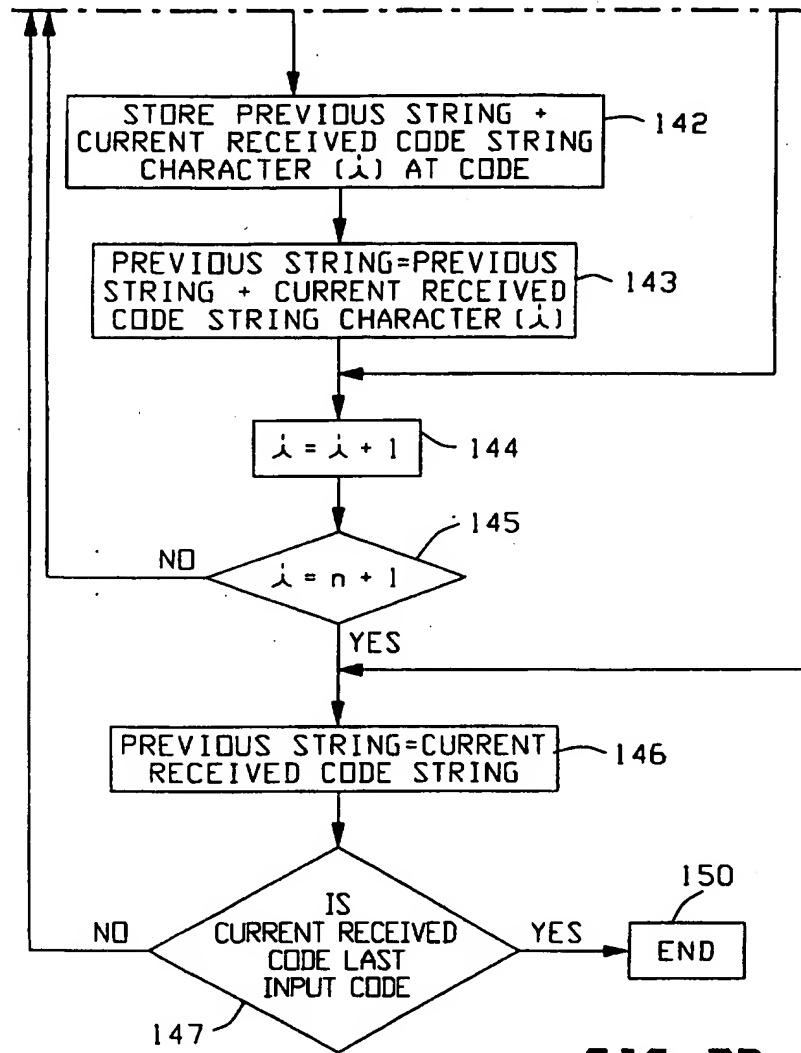


FIG. 7B

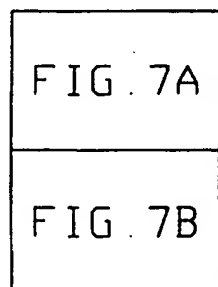


FIG. 7

8/16

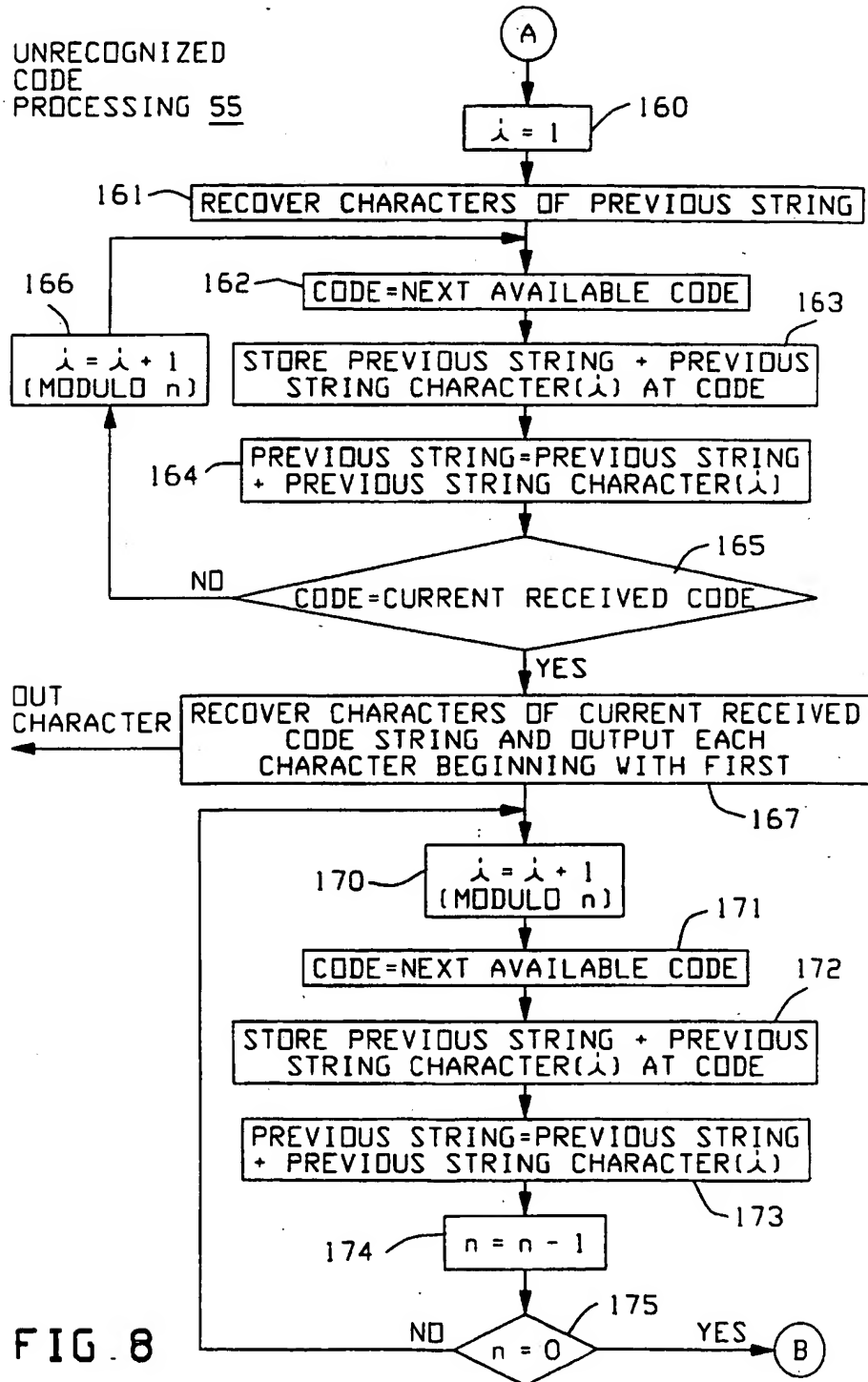
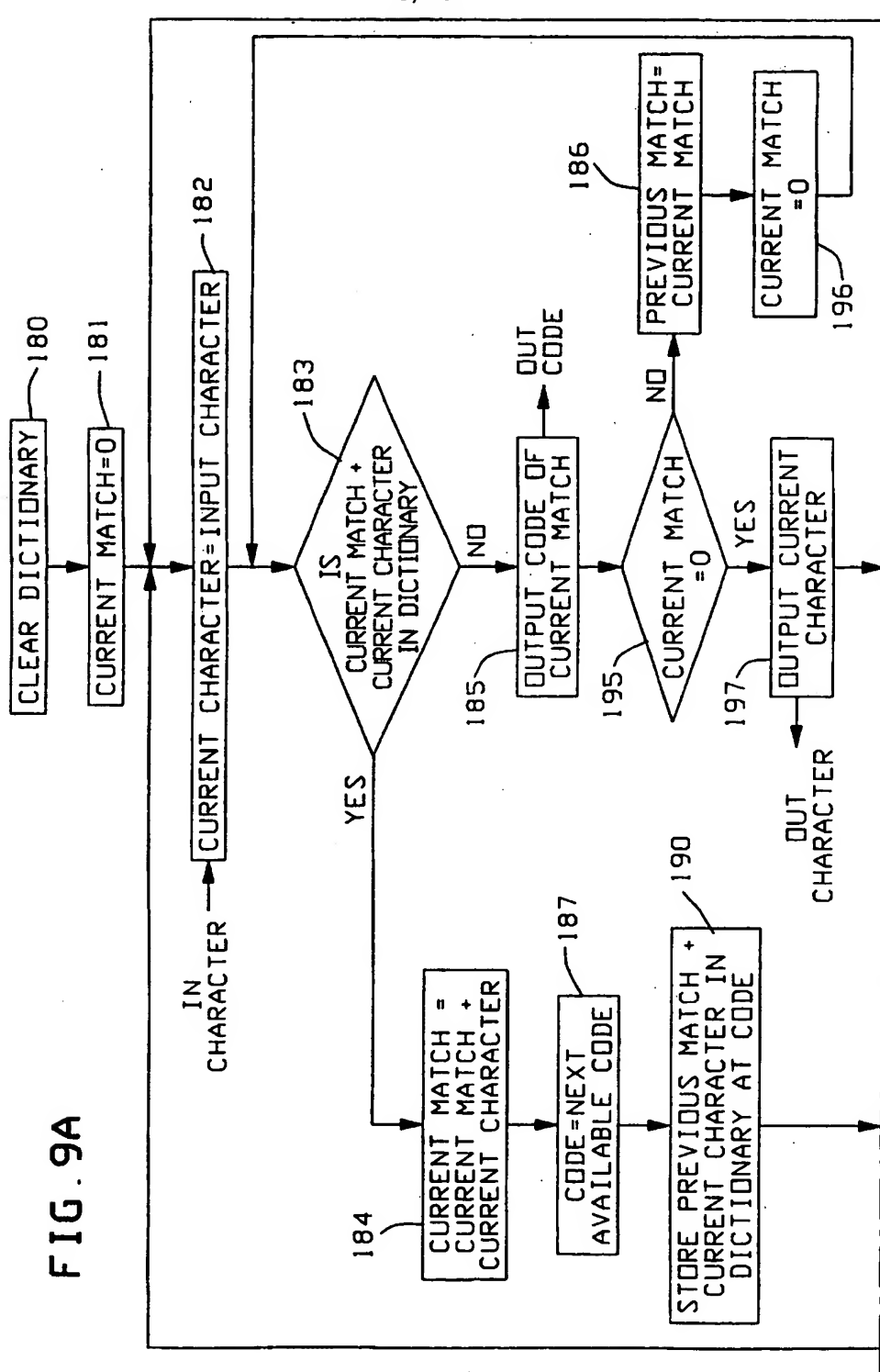
UNRECOGNIZED
CODE
PROCESSING 55

FIG. 8

9/16



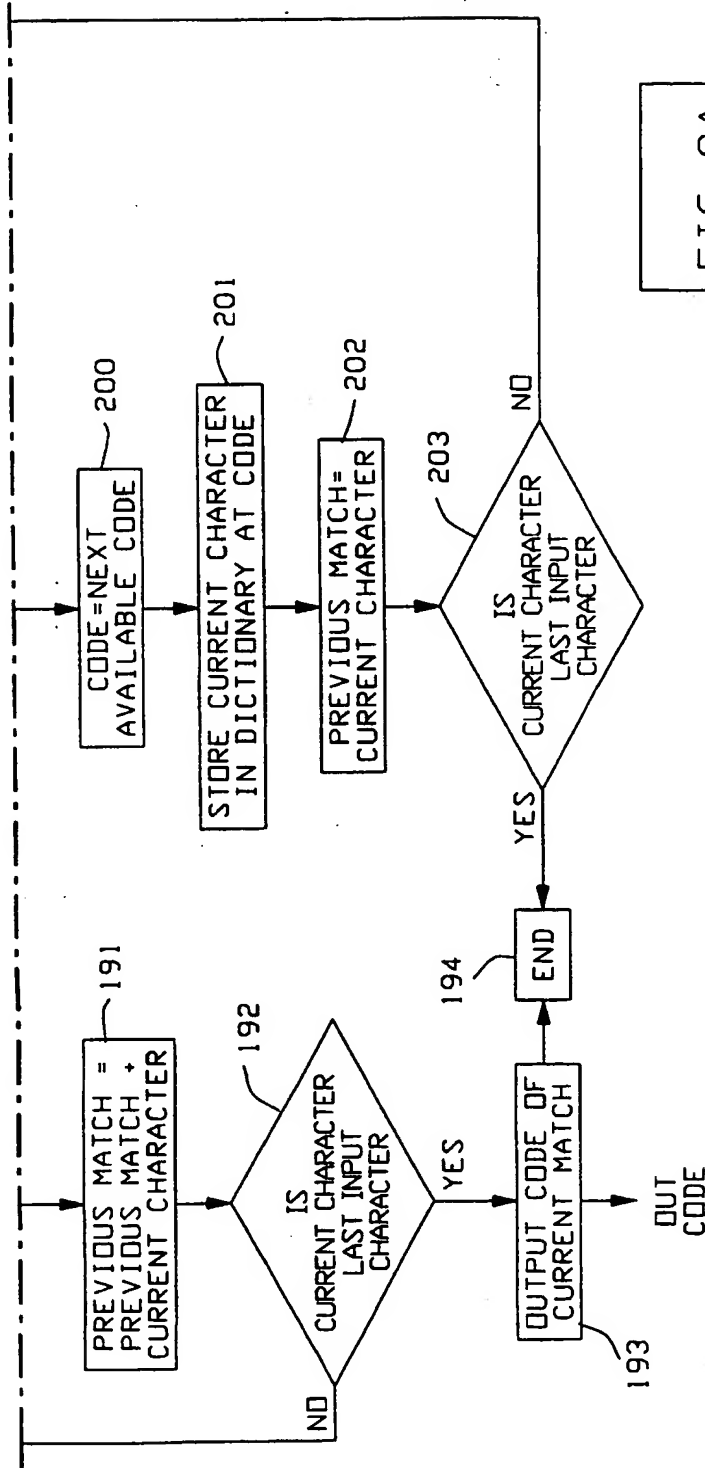


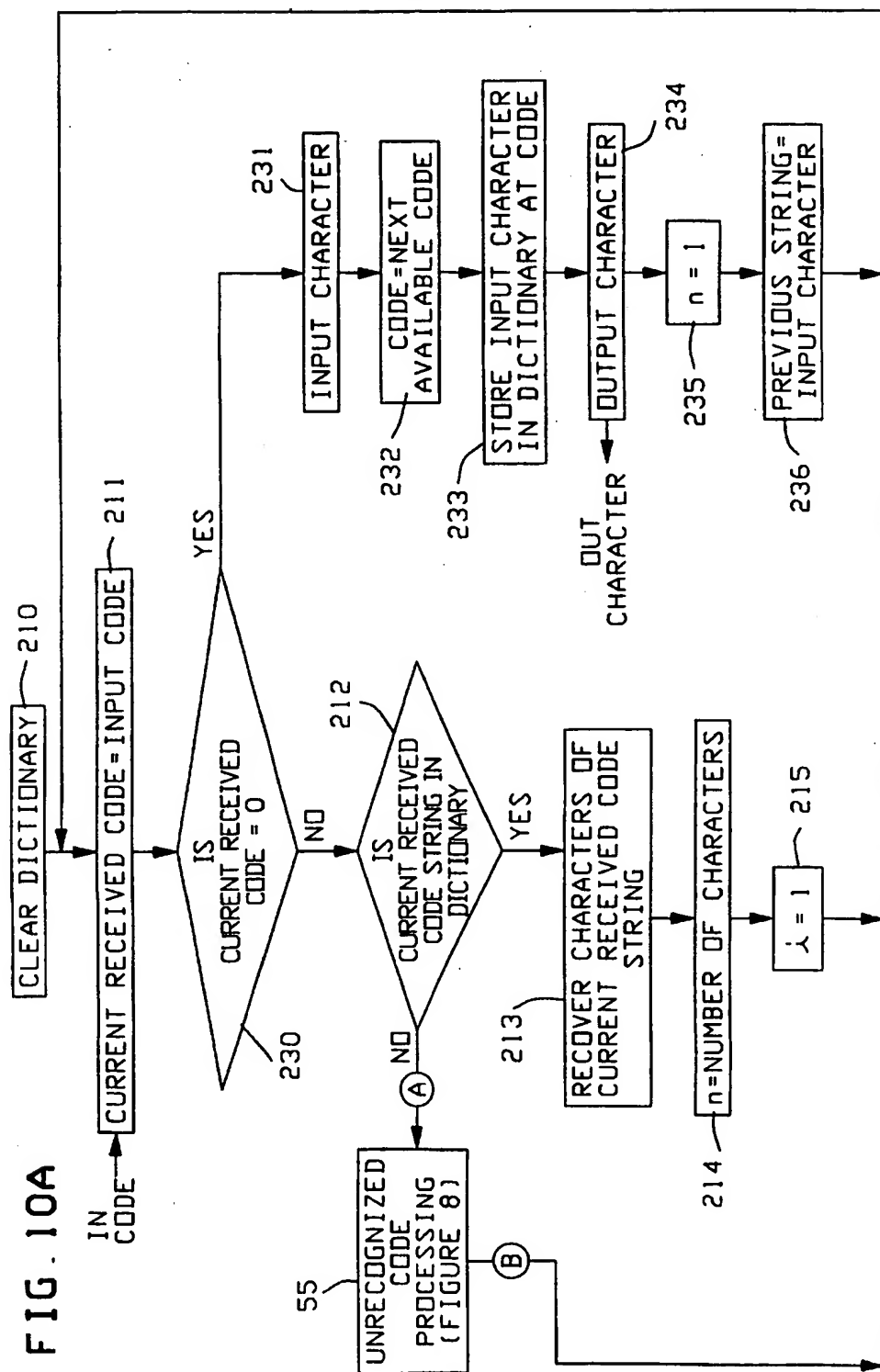
FIG. 9B

FIG. 9

FIG. 9A

FIG. 9B

FIG. 10A



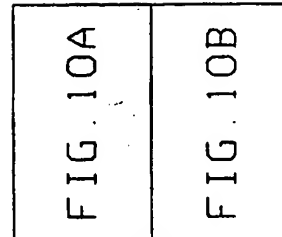
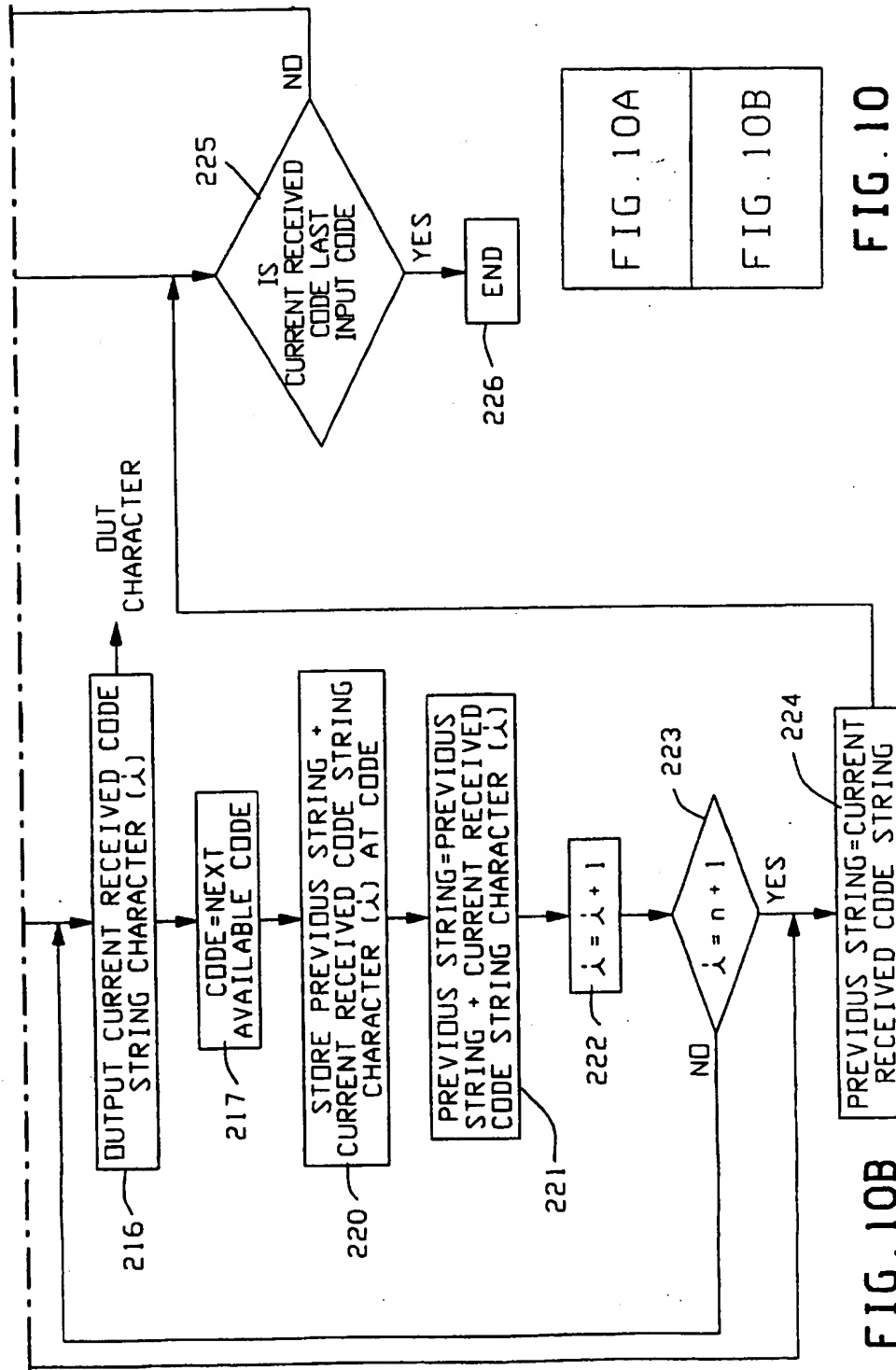


FIG. 10

13/16

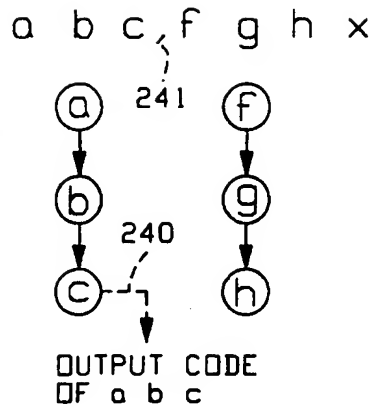


FIG. 11a

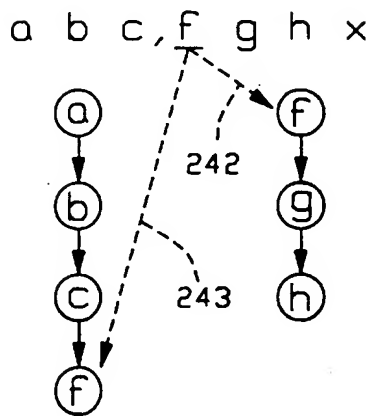


FIG. 11b

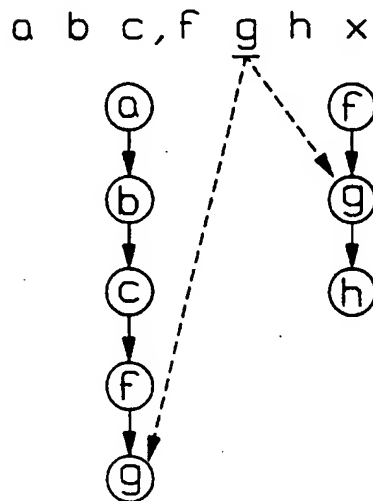


FIG. 11c

14/16

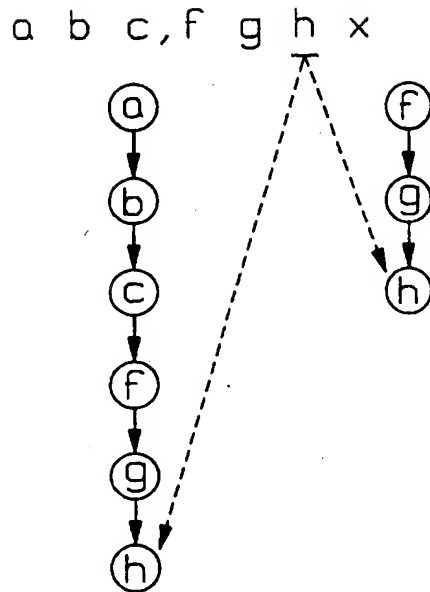


FIG. 11d

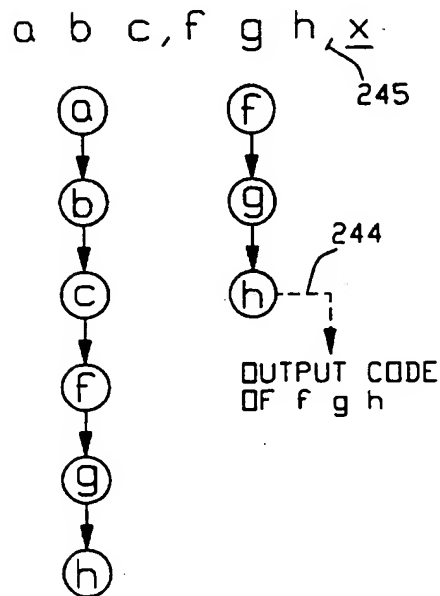


FIG. 11e

15/16

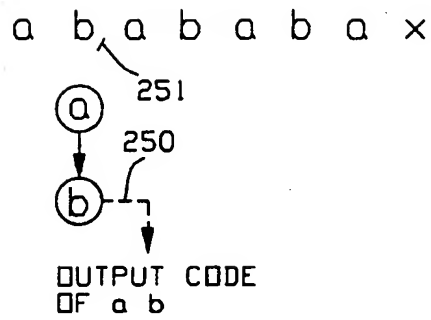


FIG. 12a

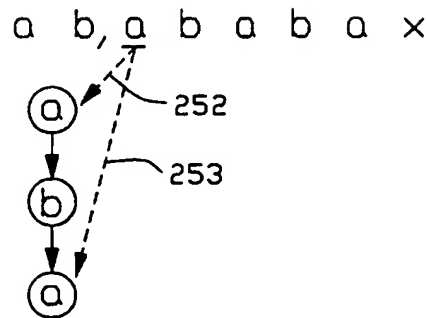


FIG. 12b

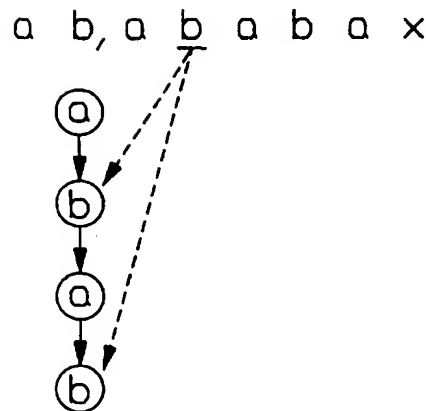


FIG. 12c

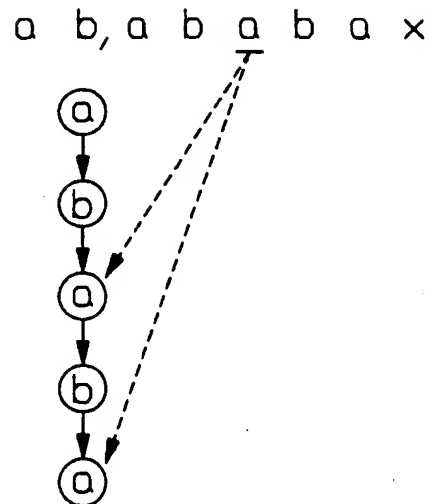
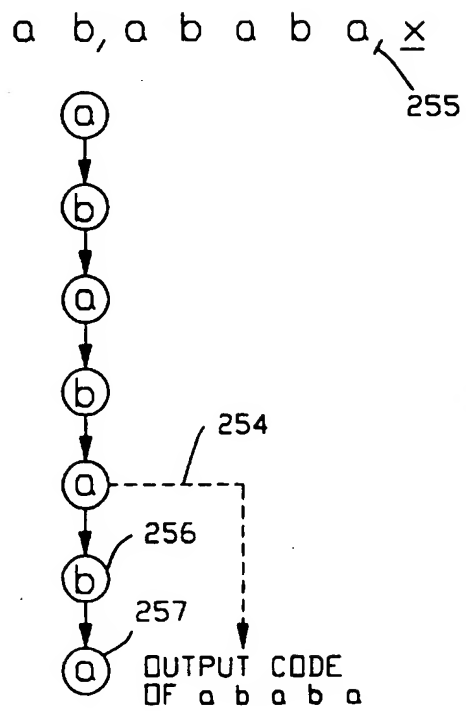
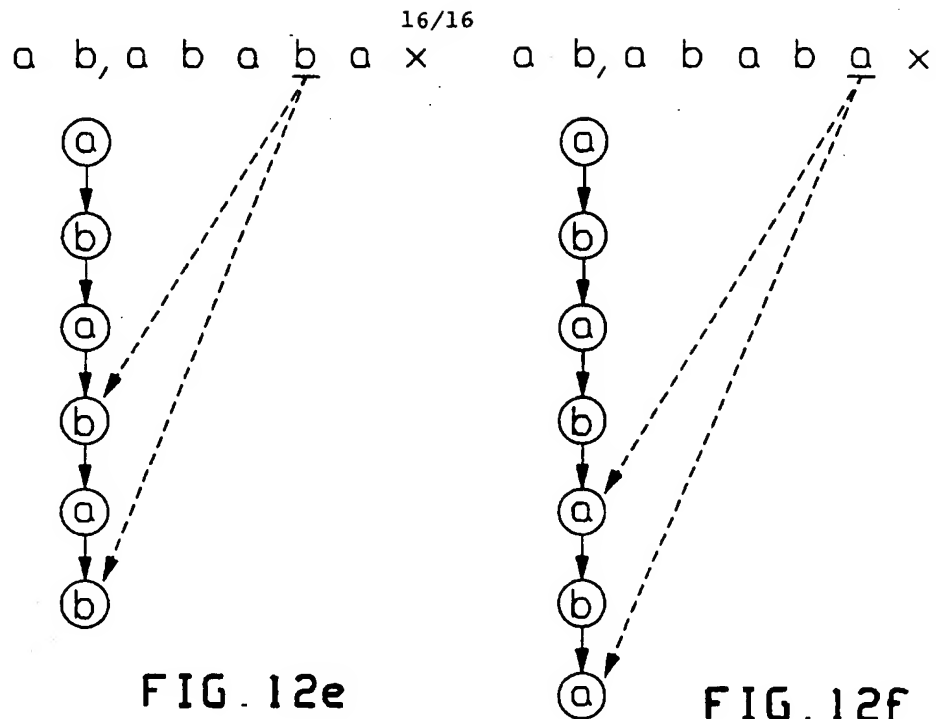


FIG. 12d



INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 97/12943

A. CLASSIFICATION OF SUBJECT MATTER IPC 6 H03M7/36		
According to International Patent Classification (IPC) or to both national classification and IPC		
B. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) IPC 6 H03M		
Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched		
Electronic data base consulted during the international search (name of data base and, where practical, search terms used)		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	GB 2 277 179 A (HEWLETT-PACKARD) 19 October 1994	1-3,7-9, 26-28, 32-34, 14,25, 39,50
A	see page 22, line 20 - page 23, line 2 see page 24, line 20 - page 25, line 25 see figures 12,13 see page 26, line 22 - page 27, line 19; figure 16 see page 28, line 20 - page 30, line 14; figure 17A	
A	US 5 253 325 A (CLARK) 12 October 1993 see column 3, line 40 - column 4, line 63	1,26
-/--		
<input checked="" type="checkbox"/> Further documents are listed in the continuation of box C.		
<input checked="" type="checkbox"/> Patent family members are listed in annex.		
* Special categories of cited documents: "A" document defining the general state of the art which is not considered to be of particular relevance "E" earlier document but published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance: the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art "Z" document member of the same patent family		
Date of the actual completion of the international search 30 October 1997		Date of mailing of the international search report 19. 11. 97
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016		Authorized officer Augarde, E

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 97/12943

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT		
Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 4 814 746 A (MILLER ET AL.) 21 March 1989 cited in the application see claim 2	1,26
A	<p>---</p> <p>NAKANO ET AL.: "Highly efficient universal coding with classifying to subdictionaries for text compression" IECE TRANSSACTIONS ON FUNDAMENTALS OF ELECTRONICS , COMMUNICATIONS AND COMPUTER SCIENCES , vol. E-77A, no. 9, September 1994, TOKYO, JP, pages 1520-1525, XP000477502 see paragraph 2.2</p> <p>-----</p>	1,26

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 97/12943

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
GB 2277179 A	19-10-94	US 5389922 A	14-02-95
		DE 4340591 A	17-11-94
		JP 7104971 A	21-04-95

US 5253325 A	12-10-93	AT 134780 T	15-03-96
		CA 2005048 A	09-06-90
		DE 68925798 D	04-04-96
		DE 68925798 T	19-09-96
		EP 0375221 A	27-06-90
		WO 9006560 A	14-06-90
		JP 4502377 T	23-04-92

US 4814746 A	21-03-89	DE 3485824 A	27-08-92
		EP 0127815 A	12-12-84
		HK 7895 A	27-01-95
		JP 1508004 C	26-07-89
		JP 59231683 A	26-12-84
		JP 63056726 B	09-11-88
		SG 173394 A	28-04-95
